

IP 帮助程序

项目 • 2023/06/12

目的

Internet 协议帮助程序 (IP 帮助程序) API 可检索和修改本地计算机的网络配置设置。

如果适用

IP 帮助程序 API 适用于以编程方式操作网络和 TCP/IP 配置有用的任何计算环境。典型的 应用程序包括 IP 路由协议和简单网络管理协议 (SNMP) 代理。

开发人员受众

IP 帮助程序 API 专为 C/C++ 程序员设计。程序员还应熟悉 Windows 网络和 TCP/IP 网络概念。

运行时要求

IP 帮助程序 API 可用于所有 Windows 平台。并非所有操作系统都支持所有函数。如果存在 Windows 套接字 2 平台限制的某些实现或功能，文档中会明确指出它们。如果在不支持该函数的平台上调用 IP 帮助程序函数，则返回ERROR_NOT_SUPPORTED。有关哪些操作系统支持特定功能的更具体信息，请参阅文档中的要求部分。

在本节中

主题	说明
IP 帮助程序中的新增功能	有关 IP 帮助程序新功能的信息。
关于 IP 帮助程序	有关面向开发人员的 IP 帮助程序编程注意事项和功能的一般信息。
使用 IP 帮助程序	与 IP 帮助程序一起使用的过程和编程技术。本部分包括基本的 IP 帮助程序编程技术，例如 入门使用 IP 帮助程序 。
IP Helper 参考	IP 帮助程序函数、结构和枚举的参考文档。

相关主题

[Windows 套接字 2](#)

[路由和远程访问服务](#)

反馈

此页面是否有帮助?

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

IP 帮助程序中的新增功能

项目 • 2023/06/13

Windows 8 和 Windows Server 2012

以下功能已添加到 Windows 8 和 Windows Server 2012 上的 IP 帮助程序 API。

一个函数，用于检索网络连接的历史带宽估计值。有关详细信息，请参阅：

- [GetIpNetworkConnectionBandwidthEstimates](#)

一个结构，包含有关可用带宽估计和由 TCP/IP 堆栈确定的相关方差的信息。有关详细信息，请参阅：

- [NL_BANDWIDTH_INFORMATION](#)

Windows 7 和 Windows Server 2008 R2

以下功能已添加到 Windows 7 和 Windows Server 2008 R2 上的 IP 帮助程序 API。

用于在以太网 MAC 地址的二进制格式和字符串格式之间转换以太网地址的函数。有关详细信息，请参阅：

- [RtlEthernetAddressToString](#)
- [RtlEthernetStringToAddress](#)

Windows Server 2008 和 Windows Vista SP1

以下函数已添加到 Windows Server 2008 和 Windows Vista 上的 IP 帮助程序 API (Service Pack 1 (SP1)) 。

适用于 IPv4 和 Internet 控制消息协议的函数 (ICMP) 。有关详细信息，请参阅：

- [Icmpsendecho2Ex](#)

Windows Vista

以下函数组已添加到 Windows Vista 及更高版本的 IP 帮助程序 API 中。

使用 IPv6 和 IPv4 进行接口转换的函数。有关详细信息，请参阅：

- [ConvertInterfaceAliasToLuid](#)

- [ConvertInterfaceGuidToLuid](#)
- [ConvertInterfaceIndexToLuid](#)
- [ConvertInterfaceLuidToGuid](#)
- [ConvertInterfaceLuidToIndex](#)
- [ConvertInterfaceLuidToNameA](#)
- [ConvertInterfaceLuidToNameW](#)
- [ConvertInterfaceNameToLuidA](#)
- [ConvertInterfaceNameToLuidW](#)
- [if_indextoname](#)
- [if_nametoindex](#)

使用 IPv6 和 IPv4 进行接口管理的函数。有关详细信息，请参阅：

- [GetIfEntry2](#)
- [GetIfStackTable](#)
- [GetIfTable2](#)
- [GetIfTable2Ex](#)
- [GetInvertedIfStackTable](#)
- [GetIpInterfaceEntry](#)
- [GetIpInterfaceTable](#)
- [InitializeIpInterfaceEntry](#)
- [SetIpInterfaceEntry](#)

使用 IPv6 和 IPv4 进行 IP 地址管理的函数。有关详细信息，请参阅：

- [CreateAnycastIpAddressEntry](#)
- [CreateUnicastIpAddressEntry](#)
- [DeleteAnycastIpAddressEntry](#)
- [DeleteUnicastIpAddressEntry](#)
- [GetAnycastIpAddressEntry](#)
- [GetAnycastIpAddressTable](#)
- [GetMulticastIpAddressEntry](#)
- [GetMulticastIpAddressTable](#)
- [GetUnicastIpAddressEntry](#)
- [GetUnicastIpAddressTable](#)
- [InitializeUnicastIpAddressEntry](#)
- [NotifyStableUnicastIpAddressTable](#)
- [SetUnicastIpAddressEntry](#)

一个与 IPv6 和 IPv4 配合使用的函数，用于 IP 表内存管理。有关详细信息，请参阅：

- [FreeMibTable](#)

使用 IPv6 和 IPv4 进行 IP 邻居地址管理的函数。有关详细信息，请参阅：

- [CreateIpNetEntry2](#)
- [DeleteIpNetEntry2](#)
- [FlushIpNetTable2](#)
- [GetIpNetEntry2](#)
- [GetIpNetTable2](#)
- [ResolveIpNetEntry2](#)
- [ResolveNeighbor](#)
- [SetIpNetEntry2](#)

适用于 IP 路径管理的 IPv6 和 IPv4 的函数。有关详细信息，请参阅：

- [FlushIpPathTable](#)
- [GetIpPathEntry](#)
- [GetIpPathTable](#)

使用 IPv6 和 IPv4 进行 IP 路由管理的函数。有关详细信息，请参阅：

- [CreateIpForwardEntry2](#)
- [DeleteIpForwardEntry2](#)
- [GetBestRoute2](#)
- [GetIpForwardEntry2](#)
- [GetIpForwardTable2](#)
- [InitializeIpForwardEntry](#)
- [SetIpForwardEntry2](#)
- [SetIpStatisticsEx](#)

用于通知的 IPv6 和 IPv4 的函数。有关详细信息，请参阅：

- [CancelMibChangeNotify2](#)
- [NotifyIpInterfaceChange](#)
- [NotifyRouteChange2](#)
- [NotifyUnicastIpAddressChange](#)

使用 IP 地址的实用工具函数。有关详细信息，请参阅：

- [ConvertIpv4MaskToLength](#)
- [ConvertLengthToIpv4Mask](#)
- [CreateSortedAddressPairs](#)
- [ParseNetworkString](#)

使用传输控制协议的函数 (TCP) 和用户数据报协议 (UDP) 检索 IPv6 或 IPv4 TCP 连接表或 UDP 侦听器表。有关详细信息，请参阅：

- [GetTcp6Table](#)
- [GetTcp6Table2](#)
- [GetTcpTable2](#)
- [GetUdp6Table](#)

使用传输控制协议的函数 (TCP) 检索连接上的扩展 TCP 统计信息。有关详细信息，请参阅：

- [GetPerTcp6ConnectionEStats](#)
- [GetPerTcpConnectionEStats](#)
- [SetPerTcp6ConnectionEStats](#)
- [SetPerTcpConnectionEStats](#)

适用于 Teredo IPv6 客户端管理的新函数。有关详细信息，请参阅：

- [GetTeredoPort](#)
- [NotifyTeredoPortChange](#)
- [NotifyStableUnicastIpAddressTable](#)

用于在 IP 地址和 IP 地址的字符串表示形式之间提供转换的实用工具函数。有关详细信息，请参阅：

- [RtlIpv4AddressToString](#)
- [RtlIpv4AddressToStringEx](#)
- [RtlIpv4StringToAddress](#)
- [RtlIpv4StringToAddressEx](#)
- [RtlIpv6AddressToString](#)
- [RtlIpv6AddressToStringEx](#)
- [RtlIpv6StringToAddress](#)
- [RtlIpv6StringToAddressEx](#)

为本地计算机上的 TCP 或 UDP 端口的连续块提供持久保留的函数。有关详细信息，请参阅：

- [CreatePersistentTcpPortReservation](#)
- [CreatePersistentUdpPortReservation](#)
- [DeletePersistentTcpPortReservation](#)
- [DeletePersistentUdpPortReservation](#)
- [LookupPersistentTcpPortReservation](#)
- [LookupPersistentUdpPortReservation](#)

以下函数已添加到 Windows Server 2003 及更高版本的 IP 帮助程序 API:

- [CancelSecurityHealthChangeNotify](#)
- [NotifySecurityHealthChange](#)

Windows XP SP2

以下函数已添加到 Windows XP 上的 IP 帮助程序 API 中, Service Pack 2 (SP2) 及更高版本:

- [GetOwnerModuleFromTcpEntry](#)
- [GetOwnerModuleFromTcp6Entry](#)
- [GetOwnerModuleFromUdpEntry](#)
- [GetOwnerModuleFromUdp6Entry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

关于 IP 帮助程序

项目 · 2023/06/13

Internet 协议帮助程序 (IP 帮助程序) 通过使应用程序能够检索有关本地计算机的网络配置的信息并修改该配置来帮助本地计算机的网络管理。IP 帮助程序还提供通知机制，以确保在本地计算机网络配置的某些方面发生更改时通知应用程序。

许多 IP 帮助程序函数传递表示与 [管理信息库](#) 技术关联的数据类型的数据类型的结构参数。IP 帮助程序函数使用这些结构来表示各种网络信息，例如 ARP 缓存条目。由于 MIB API 也使用这些结构，因此管理 [信息库参考](#) 中介绍了这些结构。尽管 IP 帮助程序 API 使用这些结构，但 IP 帮助程序不同于管理信息库 (MIB) 和简单网络管理协议 (SNMP)。

IP 帮助程序文档广泛使用术语“适配器”和“接口”。“适配器”是旧术语，是网络适配器的缩写形式，最初指的是某种形式的网络硬件。适配器是数据链接级别的抽象。

“接口”是 IETF RFC 文档中使用的较新术语。RFC 将接口定义为表示节点对连接的附件的抽象概念。接口是 IP 级抽象。

适配器和接口术语在 IP 帮助程序文档中可以互换使用。在 IP 帮助程序文档中，适配器列表可能包括软件 WAN 接口和环回接口，(这两者都不是指硬件设备)。在 IP 帮助程序 API 中，有适配器到接口的一对一映射。

IP 帮助程序提供以下方面的功能：

- [检索有关网络配置的信息](#)
- [管理网络适配器](#)
- [管理接口](#)
- [管理 IP 地址](#)
- [使用地址解析协议](#)
- [检索有关 Internet 协议和 Internet 控制消息协议的信息](#)
- [管理路由](#)
- [接收网络事件的通知](#)
- [检索有关传输控制协议和用户数据报协议的信息](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

检索有关网络配置的信息

项目 • 2023/06/13

IP 帮助程序提供有关本地计算机的网络配置的信息。 若要检索常规配置信息，请使用 [GetNetworkParams](#) 函数。 此函数返回不特定于特定适配器或接口的信息。 例如，[GetNetworkParams](#) 返回本地计算机使用的 DNS 服务器的列表。

- 有关涉及 [GetNetworkParams 的代码示例](#)，请参阅 [使用 GetNetworkParams 检索信息](#)。

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

管理网络适配器

项目 • 2023/06/12

IP 帮助程序提供用于管理网络适配器的功能。给定计算机上的接口和适配器之间存在一对一对的对应关系。接口是 IP 级抽象，而适配器是数据链接级抽象。

使用以下段落中所述的函数检索有关本地计算机中的网络适配器的信息。

[GetAdaptersInfo](#) 函数返回 `IP_ADAPTER_INFO` 结构的数组，本地计算机中的每个适配器对应一个。[GetPerAdapterInfo](#) 函数返回有关特定适配器的其他信息。

[GetPerAdapterInfo](#) 函数要求调用方指定适配器的索引。若要从适配器名称获取适配器索引，请使用 [GetAdapterIndex](#) 函数。

某些应用程序使用接收数据报的适配器，但无法传输数据报。若要获取有关这些适配器的信息，请使用 [GetUniDirectionalAdapterInfo](#) 函数。

[GetAdaptersAddresses](#) 函数使你能够检索与特定适配器关联的 IP 地址。此函数支持 IPv4 和 IPv6 寻址。

- 有关涉及 [GetAdaptersInfo 的](#) 代码示例，请参阅 [使用 GetAdaptersInfo 管理网络适配器](#)。

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

管理接口

项目 • 2023/06/13

IP 帮助程序扩展了管理网络接口的能力。给定计算机上的接口和适配器之间存在一对一对的对应关系。接口是 IP 级抽象，而适配器是数据链接级抽象。

使用以下段落中所述的函数管理本地计算机上的接口。

[GetNumberOfInterfaces](#) 函数返回本地计算机上的接口数。

[GetInterfaceInfo](#) 函数返回一个表，其中包含本地计算机上接口的名称和相应的索引。有关涉及 [GetInterfaceInfo](#) 的代码示例，请参阅 [使用 GetInterfaceInfo 管理接口](#)。

[GetFriendlyIfIndex](#) 函数采用接口索引并返回向后兼容的接口索引，即仅使用较低 24 位的接口索引。这种类型的索引有时称为“友好”接口索引。

[GetIfEntry](#) 函数返回一个 [MIB_IFROW](#) 结构，其中包含有关本地计算机上特定接口的信息。此函数要求调用方提供接口的索引。

[GetIfTable](#) 函数返回一个包含 [MIB_IFROW](#) 项的表，计算机上每个接口对应一个条目。

使用 [SetIfEntry](#) 函数修改特定接口的配置。

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

管理 IP 地址

项目 • 2023/06/13

IP 帮助程序可以帮助管理与本地计算机上的接口关联的 IP 地址。 使用以下段落中所述的函数进行 IP 地址管理。

[GetIpAddrTable](#) 函数检索包含 IP 地址到接口的映射的表。 多个 IP 地址可能与同一接口相关联。

使用 [AddIPAddress](#) 函数将 IP 地址添加到特定接口。 若要删除以前使用 [AddIPAddress](#) 添加的 IP 地址，请使用 [DeleteIPAddress](#) 函数。

[IpReleaseAddress](#) 和 [IpRenewAddress](#) 函数要求本地计算机使用动态主机配置协议 (DHCP)。 [IpReleaseAddress](#) 函数释放以前从 DHCP 获取的 IP 地址。 [IpRenewAddress](#) 函数续订特定 IP 地址上的 DHCP 租约。 DHCP 租约允许计算机在指定时间段内使用 IP 地址。

- 有关涉及 [GetIpAddrTable](#) 的代码示例，请参阅 [使用 GetIpAddrTable 管理 IP 地址](#)。
- 有关涉及 [AddIPAddress](#) 和 [DeleteIPAddress](#) 的代码示例，请参阅 [使用 AddIPAddress 和 DeleteIPAddress 管理 IP 地址](#)。
- 有关涉及 [IpReleaseAddress](#) 和 [IpRenewAddress](#) 的代码示例，请参阅 [使用 IpReleaseAddress 和 IpRenewAddress 管理 DHCP 租约](#)。

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

使用地址解析协议

项目 • 2023/06/13

可以使用 IP 帮助程序对本地计算机执行地址解析协议 (ARP) 操作。 使用以下函数检索和修改 ARP 表。

[GetIpNetTable](#) 检索 ARP 表。 ARP 表包含 IP 地址到物理地址的映射。 物理地址有时称为媒体访问控制器 (MAC) 地址。

使用 [CreateIpNetEntry](#) 和 [DeleteIpNetEntry](#) 函数向表添加或删除特定的 ARP 条目。
[FlushIpNetTable](#) 函数从表中删除所有条目。

若要创建或删除代理 ARP 条目，请使用 [CreateProxyArpEntry](#) 和 [DeleteProxyArpEntry](#) 函数。

[SendARP](#) 函数将 ARP 请求发送到本地网络。

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

检索有关 Internet 协议和 Internet 控制消息协议的信息

项目 • 2023/06/12

IP 帮助程序提供可用于本地计算机网络管理的信息检索功能。以下函数检索 Internet 协议 (IP) 和 Internet 控制消息协议 (ICMP) 的统计信息。还可以使用这些函数为 IP 设置某些配置参数。

[GetIpStatistics](#) 函数检索本地计算机的当前 IP 统计信息。有关涉及 [GetIpStatistics 的代码示例](#)，请参阅 [使用 GetIpStatistics 检索信息](#)。

[GetIcmpStatistics](#) 函数检索当前的 ICMP 统计信息。

使用 [SetIpStatistics](#) 函数启用或禁用 IP 转发。此函数还可用于设置 IP 数据报的默认生存时间 (TTL)。或者，可以使用 [SetIpTTL](#) 函数设置 TTL。

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

管理路由

项目 • 2023/06/12

IP 帮助程序提供用于管理网络路由的功能。 使用以下函数管理 IP 路由表，并获取其他路由信息。

通过调用 [GetIpForwardTable](#) 函数来检索 IP 路由表的内容。 使用 [CreateIpForwardEntry](#)、[DeleteIpForwardEntry](#) 和 [SetIpForwardEntry](#) 函数操作 IP 路由表中的特定条目。 使用 [CreateIpForwardEntry](#) 函数添加新的路由表条目。 使用 [DeleteIpForwardEntry](#) 函数删除现有条目。 [SetIpForwardEntry](#) 函数修改现有条目。

IP 帮助程序的路由器管理功能可用于检索有关如何通过网络路由数据报的信息。

[GetBestRoute](#) 函数检索到指定目标地址的最佳路由。 [GetBestInterface](#) 函数检索到指定目标地址的最佳路由所使用的接口的索引。 最后，[GetRTTAndHopCount](#) 函数检索 RTT 和指定目标地址的跃点数 (往返时间)。

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

接收网络事件的通知

项目 • 2024/01/28

使用以下函数来确保应用程序接收某些网络事件的通知。

使用 [NotifyAddrChange](#) 函数请求通知本地计算机上的 IP 地址和接口之间的映射中发生的任何更改。

同样，[NotifyRouteChange](#) 函数允许应用程序请求通知 IP 路由表中发生的任何更改。

这些函数提供的通知不指定更改的内容；它们只是指定更改了某些内容。使用其他 IP 帮助程序函数（如 [GetIPAddrTable](#) 和 [GetBestRoute](#)）来确定更改的确切性质。

反馈

此页面是否有帮助？

是

否

检索有关传输控制协议和用户数据报协议的信息

项目 • 2023/06/12

使用 IP 帮助程序可以访问有关本地计算机上使用的网络协议的信息。 使用以下段落中所述的函数检索有关本地计算机上的传输控制协议 (TCP) 和用户数据报协议 (UDP) 的信息。

[GetTcpStatistics](#) 函数检索 TCP 的当前统计信息。 有关涉及 **GetTcpStatistics 的代码示例**，请参阅 [使用 GetTcpStatistics 检索信息](#)。

[GetUdpStatistics](#) 函数检索 UDP 的当前统计信息。

[GetTcpTable](#) 函数检索 TCP 连接表。 [GetUdpTable](#) 检索 UDP 侦听器表。

[SetTcpEntry](#) 函数使开发人员能够将指定 TCP 连接的状态设置为 MIB_TCP_STATE_DELETE_TCB。

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

使用 IP 帮助程序

项目 • 2023/06/13

本部分介绍 IP 帮助程序采用的过程和编程技术。它包括基本的 IP 帮助程序编程技术，例如[使用 IP 帮助程序入门](#)，以及对更有经验的 IP 帮助程序开发人员有用的高级技术。

以下列表介绍了本节中的主题：

[使用 IP 帮助程序入门](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

使用 IP 帮助程序入门

项目 • 2023/06/13

下面是使用 IP 帮助程序应用程序编程接口 (API) 开始编程的分步指南。 它旨在了解基本的 IP 帮助程序函数和数据结构，以及它们如何协同工作。

用于演示的应用程序是一个非常基本的 IP 帮助程序应用程序。 Microsoft Windows 软件开发工具包 (SDK) 附带的示例中包含了更高级的代码示例。

对于大多数 IP 帮助程序应用程序，第一步相同。

- [创建基本 IP 帮助程序应用程序](#)

以下部分介绍创建此基本 IP 帮助程序应用程序的剩余步骤。

- [使用 GetNetworkParams 检索信息](#)
- [使用 GetAdaptersInfo 管理网络适配器](#)
- [使用 GetInterfaceInfo 管理接口](#)
- [使用 GetIpAddrTable 管理 IP 地址](#)
- [使用 IpReleaseAddress 和 IpRenewAddress 管理 DHCP 租约](#)
- [使用 AddIPAddress 和 DeleteIPAddress 管理 IP 地址](#)
- [使用 GetIpStatistics 检索信息](#)
- [使用 GetTcpStatistics 检索信息](#)

此基本 IP 帮助程序示例的完整源代码。

- [完整的 IP 帮助程序应用程序源代码](#)

高级 IP 帮助程序示例

Microsoft Windows 软件开发工具包 (SDK) 附带了几个更高级的 IP 帮助程序示例。 默认情况下，IP 帮助程序示例源代码由适用于 Windows 7 的 Windows SDK 安装在以下目录中：

C:\Program Files\Microsoft SDKs\Windows\v7.0\Samples\NetDs\IPHelp

下面列出的更高级示例在以下目录中找到：

- EnableRouter

此目录包含一个示例，演示如何使用 [EnableRouter](#) 和 [UnenableRouter](#) IP 帮助程序函数在本地计算机上启用和禁用 IPv4 转发。

- iparp

此目录包含一个示例程序，该程序演示如何使用 IP 帮助程序函数显示和操作本地计算机上的 IPv4 ARP 表中的条目。

- ipchange

此目录包含一个示例程序，该程序演示如何使用 IP 帮助程序函数以编程方式更改计算机上的特定网络适配器的 IP 地址。此程序还演示如何检索现有的网络适配器 IP 配置信息。

- IPConfig

此目录包含一个示例程序，该程序演示如何以编程方式检索与 IPCONFIG.EXE 实用工具类似的 IPv4 配置信息。它演示如何使用 [GetNetworkParams](#) 和 [GetAdaptersInfo](#) 函数。请注意，[GetAdaptersInfo](#) 函数仅检索 IPv4 信息。

- IPRenew

此目录包含一个示例程序，该程序演示如何以编程方式释放和续订通过 DHCP 获取的 IPv4 地址。此程序还演示了如何检索现有的网络适配器配置信息。

- IPRoute

此目录包含一个示例程序，该程序演示如何使用 IP 帮助程序函数操作 IPv4 路由表。

- ipstat

此目录包含一个示例程序，该程序演示如何使用 IP 帮助程序函数显示协议的 IPv4 连接。默认情况下，会显示 IP、ICMP、TCP 和 UDP 的统计信息。

- Netinfo

此目录包含一个示例程序，演示如何使用 Windows Vista 和更高版本上引入的新 IP 帮助程序 API 来显示/更改 IPv4 和 IPv6 的地址和接口信息。

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

创建基本 IP 帮助程序应用程序

项目 • 2023/06/13

创建基本 IP 帮助程序应用程序

1. 创建新的空项目。
2. 将空的 C++ 源文件添加到项目。
3. 确保生成环境引用平台软件开发工具包 (SDK) 的 Include、Lib 和 Src 目录。
4. 确保生成环境链接到 IP 帮助程序库文件 `Iphlpapi.lib` 和 Winsock 库文件 `WS2_32.lib`。

① 备注

某些基本 Winsock 函数用于返回 IP 地址值和其他信息。

5. 开始对 IP 帮助程序应用程序进行编程。通过包含 IP 帮助程序头文件，使用 IP 帮助程序 API。

C++

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>

int main() {
    return 0;
}
```

① 备注

使用 IP 帮助程序函数的应用程序需要 `Iphlpapi.h` 头文件。`Iphlpapi.h` 头文件自动包含其他标头文件，其中包含 IP 帮助程序函数使用的结构和枚举。

Windows Vista 和更高版本中引入的新 IP 帮助程序函数在 `Netioapi.h` 头文件中定义，该文件由 `Iphlpapi.h` 头文件自动包含。永远不应直接使用 `Netioapi.h` 头文件。

IP 帮助程序函数使用的许多结构和枚举在 *Iprtrmib.h*、*Ipexport.h* 和 *Iptypes.h* 头文件中定义。这些头文件会自动包含在 *Iphlpapi.h* 头文件中，永远不应直接使用。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK)，头文件的组织已更改。某些结构现在在 *Ipmib.h*、*Tcpmib.h* 和 *Udpmib.h* 头文件中定义，而不是在 *Iprtrmib.h* 头文件中定义。*Ipmib.h* 头文件自动包含 *Ifmib.h* 头文件。请注意，这些头文件会自动包含在 *Iphlpapi.h* 头文件中的 *Iprtrmib.h* 中。

大多数使用 IP 帮助程序 API 的应用程序都需要 Windows 套接字 2.0 的 *Winsock2.h* 头文件。如果需要 *Winsock2.h* 头文件，则应将此文件的 #include 行放在 *Iphlpapi.h* 头文件的 #include 行之前。

Winsock2.h 头文件在内部包含 *Windows.h* 头文件中的核心元素，因此 IP 帮助程序应用程序中通常没有 *windows.h* 头文件 #include 行。如果 *Windows.h* 头文件需要 #include 行，则前面应包含#define WIN32_LEAN_AND_MEAN 宏。出于历史原因，*Windows.h* 标头默认包含 Windows 套接字 1.1 的 *Winsock.h* 头文件。Windows 套接字 1.1 的 *Winsock.h* 头文件中的声明将与 Windows 套接字 2.0 所需的 *Winsock2.h* 头文件中的声明冲突。WIN32_LEAN_AND_MEAN 宏可防止 *Windows.h* 头文件包含 *Winsock.h* 头文件。下面显示了说明这一点的示例。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>

#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>

int main() {
    return 0;
}
```

① 备注

此基本 IP 帮助程序应用程序仅使用 Windows 套接字 2.0 中的某些 IP 地址数据结构和 IP 地址到字符串转换函数。 使用这些资源后，无需调用 **WSAStartup** 即可使用这些 Windows 套接字函数来初始化 Windows 套接字资源和 **WSACleanup**。

在使用这些 IP 地址以外的其他 Winsock 函数到字符串函数的 IP 帮助程序应用程序中，必须先调用 **WSAStartup** 函数来初始化 Windows 套接字资源，然后才能调用任何 Windows 套接字函数，并在应用程序使用 Windows 套接字资源时调用 **WSACleanup**。

① 备注

Stdio.h 头文件是从此基本 IP 帮助程序应用程序中使用各种标准 C 函数所必需的。

下一步： [使用 GetNetworkParams 检索信息](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

使用 GetNetworkParams 检索信息

项目 • 2023/06/13

[GetNetworkParams](#) 函数使用有关当前网络设置的数据填充指向[FIXED_INFO](#)结构的指针。

使用 GetNetworkParams

1. 声明指向名为 *pFixedInfo* 的[FIXED_INFO](#)对象的指针，以及名为 *ulOutBufLen* 的 [ULONG](#) 对象。这些变量作为参数传递给 [GetNetworkParams](#) 函数。此外，创建用于) 进行错误检查的 [DWORD](#) 变量 *dwRetVal*。

C++

```
FIXED_INFO *pFixedInfo;  
IP_ADDR_STRING *pIPAddr;  
  
ULONG ulOutBufLen;  
DWORD dwRetVal;
```

2. 为结构分配内存。

① 备注

ulOutBufLen 的大小不足以保存信息。查看下一步。

C++

```
pFixedInfo = (FIXED_INFO *) malloc(sizeof(FIXED_INFO));  
ulOutBufLen = sizeof(FIXED_INFO);
```

3. 对 [GetNetworkParams](#) 进行初始调用，以获取 *ulOutBufLen* 变量所需的大小。

① 备注

此函数将失败，用于确保 *ulOutBufLen* 变量指定的大小足以保存返回给 *pFixedInfo* 的所有数据。这是此类型的数据结构和函数的常见编程模型。

C++

```
if (GetNetworkParams(pFixedInfo, &ulOutBufLen) ==  
    ERROR_BUFFER_OVERFLOW) {  
    free(pFixedInfo);  
    pFixedInfo = (FIXED_INFO *) malloc(ulOutBufLen);  
    if (pFixedInfo == NULL) {  
        printf("Error allocating memory needed to call  
GetNetworkParams\n");  
    }  
}
```

4. 使用常规错误检查并将其值返回到 **DWORD** 变量 *dwRetVal*, 对 **GetNetworkParams** 进行第二次调用;用于更高级的错误检查。

C++

```
if (dwRetVal = GetNetworkParams(pFixedInfo, &ulOutBufLen) !=  
    NO_ERROR) {  
    printf("GetNetworkParams failed with error %d\n", dwRetVal);  
    if (pFixedInfo) {  
        free(pFixedInfo);  
    }  
}
```

5. 如果调用成功, 请从 *pFixedInfo* 数据结构访问数据。

C++

```
printf("\tHost Name: %s\n", pFixedInfo->HostName);  
printf("\tDomain Name: %s\n", pFixedInfo->DomainName);  
printf("\tDNS Servers:\n");  
printf("\t\t%s\n", pFixedInfo->DnsServerListIpAddress.String);  
  
pIPAddr = pFixedInfo->DnsServerList.Next;  
while (pIPAddr) {  
    printf("\t\t%s\n", pIPAddr->IpAddress.String);  
    pIPAddr = pIPAddr->Next;  
}  
  
printf("\tNode Type: ");  
switch (pFixedInfo->NodeType) {  
case 1:  
    printf("%s\n", "Broadcast");  
    break;  
case 2:  
    printf("%s\n", "Peer to peer");  
    break;  
case 4:  
    printf("%s\n", "Mixed");  
    break;
```

```
    case 8:
        printf("%s\n", "Hybrid");
        break;
    default:
        printf("\n");
    }

    printf("\tNetBIOS Scope ID: %s\n", pFixedInfo->ScopeId);

    if (pFixedInfo->EnableRouting)
        printf("\tIP Routing Enabled: Yes\n");
    else
        printf("\tIP Routing Enabled: No\n");

    if (pFixedInfo->EnableProxy)
        printf("\tWINS Proxy Enabled: Yes\n");
    else
        printf("\tWINS Proxy Enabled: No\n");

    if (pFixedInfo->EnableDns)
        printf("\tNetBIOS Resolution Uses DNS: Yes\n");
    else
        printf("\tNetBIOS Resolution Uses DNS: No\n");
```

6. 释放为 *pFixedInfo* 结构分配的任何内存。

C++

```
if (pFixedInfo) {
    free(pFixedInfo);
    pFixedInfo = NULL;
}
```

下一步： [使用 GetAdaptersInfo 管理网络适配器](#)

上一步： [创建基本 IP 帮助程序应用程序](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

使用 GetAdaptersInfo 管理网络适配器

项目 • 2023/06/13

[GetAdaptersInfo](#) 函数使用与系统关联的网络适配器的相关信息填充指向 [IP_ADAPTER_INFO](#) 结构的指针。

使用 GetAdaptersInfo

1. 声明指向名为 *pAdapterInfo* 的 [IP_ADAPTER_INFO](#) 变量的指针，以及一个名为 *ulOutBufLen* 的 [ULONG](#) 变量。这些变量作为参数传递给 [GetAdaptersInfo](#) 函数。此外，创建名为 *dwRetVal* (的 [DWORD](#) 变量，用于) 进行错误检查。

C++

```
IP_ADAPTER_INFO *pAdapterInfo;
ULONG           ulOutBufLen;
DWORD           dwRetVal;
```

2. 为结构分配内存。

C++

```
pAdapterInfo = (IP_ADAPTER_INFO *) malloc( sizeof(IP_ADAPTER_INFO) );
ulOutBufLen = sizeof(IP_ADAPTER_INFO);
```

3. 对 [GetAdaptersInfo](#) 进行初始调用，以获取 *ulOutBufLen* 变量中所需的大小。

① 备注

此对函数的调用意味着失败，并且用于确保 *ulOutBufLen* 变量指定的大小足以保存返回给 *pAdapterInfo* 的所有信息。这是此类型的数据结构和函数的常见编程模型。

C++

```
if (GetAdaptersInfo( pAdapterInfo, &ulOutBufLen ) != ERROR_SUCCESS) {
    free (pAdapterInfo);
    pAdapterInfo = (IP_ADAPTER_INFO *) malloc ( ulOutBufLen );
}
```

4. 再次调用 **GetAdaptersInfo**，将 *pAdapterInfo* 和 *ulOutBufLen* 作为参数传递并执行常规错误检查。将其值返回到 DWORD 变量 *dwRetVal* (以便) 进行更广泛的错误检查。

C++

```
if ((dwRetVal = GetAdaptersInfo( pAdapterInfo, &ulOutBufLen)) !=  
    ERROR_SUCCESS) {  
    printf("GetAdaptersInfo call failed with %d\n", dwRetVal);  
}
```

5. 如果调用成功，请访问 *pAdapterInfo* 结构中的某些数据。

C++

```
PIP_ADAPTER_INFO pAdapter = pAdapterInfo;  
while (pAdapter) {  
    printf("Adapter Name: %s\n", pAdapter->AdapterName);  
    printf("Adapter Desc: %s\n", pAdapter->Description);  
    printf("\tAdapter Addr: \t");  
    for (UINT i = 0; i < pAdapter->AddressLength; i++) {  
        if (i == (pAdapter->AddressLength - 1))  
            printf("%.2X\n", (int)pAdapter->Address[i]);  
        else  
            printf("%.2X-", (int)pAdapter->Address[i]);  
    }  
    printf("IP Address: %s\n", pAdapter->IpAddressList.IpAddress.String);  
    printf("IP Mask: %s\n", pAdapter->IpAddressList.IpMask.String);  
    printf("\tGateway: \t%s\n", pAdapter->GatewayList.IpAddress.String);  
    printf("\t***\n");  
    if (pAdapter->DhcpEnabled) {  
        printf("\tDHCP Enabled: Yes\n");  
        printf("\t\tDHCP Server: \t%s\n", pAdapter->DhcpServer.IpAddress.String);  
    }  
    else  
        printf("\tDHCP Enabled: No\n");  
  
    pAdapter = pAdapter->Next;  
}
```

6. 释放为 *pAdapterInfo* 结构分配的任何内存。

C++

```
if (pAdapterInfo)
    free(pAdapterInfo);
```

下一步： [使用 GetInterfaceInfo 管理接口](#)

上一步： [使用 GetNetworkParams 检索信息](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

使用 GetInterfaceInfo 管理接口

项目 • 2023/06/13

[GetInterfaceInfo](#) 函数使用与系统关联的接口的相关信息填充指向 `IP_INTERFACE_INFO` 结构的指针。

使用 GetInterfaceInfo

1. 声明指向名为 `pInfo` 的 `IP_INTERFACE_INFO` 对象的指针和名为 `ulOutBufLen` 的 `ULONG` 对象。此外，声明一个名为 `dwRetVal` (`DWORD` 对象，用于) 进行错误检查。

C++

```
ULONG          ulOutBufLen;
DWORD          dwRetVal;
unsigned int   i;

IP_INTERFACE_INFO* pInterfaceInfo;
```

2. 为结构分配内存。

⚠ 备注

的大小 `ulOutBufLen` 不足以保存信息。查看下一步。

C++

```
pInterfaceInfo = (IP_INTERFACE_INFO *) malloc(sizeof(IP_INTERFACE_INFO));
ulOutBufLen = sizeof(IP_INTERFACE_INFO);
```

3. 对 [GetInterfaceInfo](#) 进行初始调用，以获取变量中 `ulOutBufLen` 所需的大小。

⚠ 备注

对函数的此调用意味着失败，用于确保 `ulOutBufLen` 变量指定的大小足以保存返回给 `pInfo` 的所有信息。这是 IP 帮助器中用于此类型的数据结构和函数的常见编程模型。

C++

```
if (GetInterfaceInfo(pInterfaceInfo, &ulOutBufLen) ==  
    ERROR_INSUFFICIENT_BUFFER) {  
    free(pInterfaceInfo);  
    pInterfaceInfo = (IP_INTERFACE_INFO *) malloc(ulOutBufLen);  
}
```

4. 使用常规错误检查对 `GetInterfaceInfo` 进行第二次调用，并将其值返回到 `DWORD` 变量 `dwRetVal` (，以便) 进行更高级的错误检查。

C++

```
if ((dwRetVal = GetInterfaceInfo(pInterfaceInfo, &ulOutBufLen)) !=  
    NO_ERROR) {  
    printf("  GetInterfaceInfo failed with error: %d\n", dwRetVal);  
}
```

5. 如果调用成功，请从 `pInfo` 数据结构访问数据。

C++

```
printf("  GetInterfaceInfo succeeded.\n");  
  
printf("  Num Adapters: %ld\n\n", pInfo->NumAdapters);  
for (i = 0; i < (unsigned int) pInfo->NumAdapters;  
i++) {  
    printf("    Adapter Index[%d]: %ld\n", i,  
          pInfo->Adapter[i].Index);  
    printf("    Adapter Name[%d]: %ws\n\n", i,  
          pInfo->Adapter[i].Name);  
}
```

① 备注

第一行中的 `%ws` 表示宽字符串。之所以使用此方法，是因为 `IP_ADAPTER_INDEX_MAP` 结构的 `Adapter Name` 属性是 `WCHAR`，它是 Unicode 字符串。

6. 释放为 `pInfo` 结构分配的任何内存。

C++

```
if (pInterfaceInfo) {  
    free(pInterfaceInfo);  
    pInterfaceInfo = NULL;  
}
```

下一步： [使用 GetIpAddrTable 管理 IP 地址](#)

上一步： [使用 GetAdaptersInfo 管理网络适配器](#)

反馈

此页面是否有帮助？



[在 Microsoft Q&A 获得帮助](#)

使用 GetIpAddrTable 管理 IP 地址

项目 • 2023/06/13

[GetIpAddrTable](#) 函数使用与系统关联的当前 IP 地址的信息填充指向[MIB_IPADDRTABLE](#)结构的指针。

使用 GetIpAddrTable

1. 声明指向名为 *pIPAddrTable* 的 [MIB_IPADDRTABLE](#) 对象的指针，以及名为 *dwSize* 的 DWORD 对象。这些变量作为参数传递给 [GetIpAddrTable](#) 函数。此外，创建名为 *dwRetVal* (的 DWORD 变量，用于) 进行错误检查。

C++

```
MIB_IPADDRTABLE *pIPAddrTable;  
DWORD dwSize = 0;  
DWORD dwRetVal;
```

2. 为结构分配内存。

① 备注

dwSize 的大小不足以保存信息。 查看下一步。

C++

```
pIPAddrTable = (MIB_IPADDRTABLE*) malloc( sizeof(MIB_IPADDRTABLE) );
```

3. 对 [GetIpAddrTable](#) 进行初始调用，以获取 *dwSize* 变量中所需的大小。

① 备注

此函数调用意味着失败，并且用于确保 *dwSize* 变量指定的大小足以保存返回给 *pIPAddrTable* 的所有信息。这是此类型的数据结构和函数的常见编程模型。

C++

```
if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==  
    ERROR_INSUFFICIENT_BUFFER) {  
    free( pIPAddrTable );  
    pIPAddrTable = (MIB_IPADDRTABLE *) malloc ( dwSize );  
}
```

4. 使用常规错误检查对 `GetIpAddrTable` 进行第二次调用，并将其值返回到 `DWORD` 变量 `dwRetVal`，以便进行更高级的错误检查。

C++

```
if ( (dwRetVal = GetIpAddrTable( pIPAddrTable, &dwSize, 0 )) !=  
    NO_ERROR ) {  
    printf("GetIpAddrTable call failed with %d\n", dwRetVal);  
}
```

5. 如果调用成功，请从 `pIPAddrTable` 数据结构访问数据。

C++

```
printf("IP Address:      %ld\n", pIPAddrTable->table[0].dwAddr);  
printf("IP Mask:         %ld\n", pIPAddrTable->table[0].dwMask);  
printf("IF Index:        %ld\n", pIPAddrTable->table[0].dwIndex);  
printf("Broadcast Addr:  %ld\n", pIPAddrTable-  
>table[0].dwBCastAddr);  
printf("Re-assembly size: %ld\n", pIPAddrTable-  
>table[0].dwReasmSize);
```

6. 释放为 `pIPAddrTable` 结构分配的任何内存。

C++

```
if (pIPAddrTable)  
    free(pIPAddrTable);
```

① 备注

`DWORD` 对象 `dwAddr` 和 `dwMask` 以主机字节顺序而不是网络字节顺序作为数字值返回。这些值不是虚线 IP 地址。

[下一步：使用 IpReleaseAddress 和 IpRenewAddress 管理 DHCP 租约](#)

[上一步：使用 GetInterfaceInfo 管理接口](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

使用 IpReleaseAddress、 IpRenewAddress 管理 DHCP 租约

项目 • 2023/06/13

[IpReleaseAddress](#) 和 [IpRenewAddress](#) 函数用于发布和续订当前动态主机配置协议 (DHCP) 租约。 [IpReleaseAddress](#) 函数释放以前通过 DHCP 获取的 IPv4 地址。

[IpRenewAddress](#) 函数续订以前通过 DHCP 获取的 IPv4 地址的租约。通常结合使用这两个函数，首先通过调用 [IpReleaseAddress](#) 释放租约，然后通过调用 [IpRenewAddress](#) 函数续订租约。

如果 DHCP 客户端以前已获取 DHCP 租约，并且未在 [IpRenewAddress](#) 函数之前调用 [IpReleaseAddress](#)，则 DHCP 客户端请求将发送到发出初始 DHCP 租约的 DHCP 服务器。此 DHCP 服务器可能不可用，或者 DHCP 请求可能会失败。如果主机之前已获取 DHCP 租约，并在 [IpRenewAddress](#) 函数之前调用 [IpReleaseAddress](#)，则 DHCP 客户端将首先释放获取的 IP 地址，并发送 DHCP 客户端请求以获取来自任何可用 DHCP 服务器的响应。

① 备注

[IpReleaseAddress](#) 和 [IpRenewAddress](#) 函数要求启用 DHCP 才能正确执行。

[IpReleaseAddress](#) 函数将指向 [IP_ADAPTER_INDEX_MAP](#) 结构的指针作为其唯一参数。若要获取此参数，请首先调用 [GetInterfaceInfo](#)。有关 [GetInterfaceInfo](#) 函数的帮助，请参阅 [使用 GetInterfaceInfo 管理接口](#)。

使用 [IpReleaseAddress](#)

1. 使用 [GetInterfaceInfo](#) 函数获取指向[IP_ADAPTER_INDEX_MAP](#)结构的指针。(有关 [GetInterfaceInfo](#) 函数的帮助，请参阅 [使用 GetInterfaceInfo 管理接口](#)。创建用于错误检查) (DWORD 对象 `dwRetVal`)。假定 [GetInterfaceInfo](#) 返回的变量名为 `pInfo`。

C++

```
DWORD dwRetVal;
```

2. 如果 DHCP 已启用, 请调用 [IpReleaseAddress](#) 函数, 并将 [IP_ADAPTER_INDEX_MAP](#) 变量 `Adapter` 作为其参数传递。检查一般错误并将其值返回到 [DWORD](#) 变量 `dwRetVal` (, 以便) 进行更广泛的错误检查。

① 备注

`GetAdaptersInfo` 函数返回一个参数, 可用于在调用这些函数之前检查 DHCP 是否已启用。有关 `GetAdaptersInfo` 的帮助, 请参阅 [使用 GetAdaptersInfo 管理网络适配器](#)。

C++

```
if ((dwRetVal = IpReleaseAddress(&pInfo->Adapter[0])) == NO_ERROR) {  
    printf("Ip Release succeeded.\n");  
}
```

① 备注

通常将这两个函数一起使用, 即调用 `IpReleaseAddress` 函数, 然后调用 `IpRenewAddress` 函数, 并将与参数相同的结构传递给这两个函数。以下过程假定函数不一起使用: 但是, 如果函数一起使用, 请跳过步骤 1。

使用 `IpRenewAddress`

1. 使用 `GetInterfaceInfo` 函数获取指向[IP_ADAPTER_INDEX_MAP](#)结构的指针。 (有关 `GetInterfaceInfo` 函数的帮助, 请参阅 [使用 GetInterfaceInfo 管理接口](#)。 声明一个 [DWORD](#) 对象 `dwRetVal` (用于错误检查) (如果尚未声明此变量)。 假定 `GetInterfaceInfo` 返回的变量名为 `pInfo`。

C++

```
DWORD dwRetVal;
```

2. 调用 `IpRenewAddress` 函数, 将 [IP_ADAPTER_INDEX_MAP](#) 变量 `Adapter` 作为其参数传递。检查一般错误并将其值返回到 [DWORD](#) 变量 `dwRetVal` (, 以便) 进行更广泛的错误检查。

C++

```
if ((dwRetVal = IpRenewAddress(&pInfo->Adapter[0])) == NO_ERROR) {  
    printf("Ip Renew succeeded.\n");  
}
```

[下一步：使用 AddIPAddress 和 DeleteIPAddress 管理 IP 地址](#)

[上一步：使用 GetIpAddrTable 管理 IP 地址](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

使用 AddIPAddress、DeleteIPAddress 管理 IP 地址

项目 · 2023/06/13

[AddIPAddress](#) 函数将指定的 IPv4 地址添加到指定的适配器。 [DeleteIPAddress](#) 函数从指定的适配器中删除指定的 IPv4 地址。 这些函数可用于向网络适配器添加和删除 IPv4 地址。

[AddIPAddress](#) 函数添加的 IPv4 地址不是永久性的。 只要适配器对象存在，IPv4 地址才存在。 重新启动计算机会破坏 IPv4 地址，手动重置网络接口卡 (NIC)。

成功调用 [AddIPAddress](#) 后，将为添加的 IP 地址禁用 DHCP。因此，需要启用 DHCP 的 [IpReleaseAddress](#) 等函数将无法在添加的 IP 地址上正常工作。[DeleteIPAddress](#) 函数可用于删除添加的 IPv4 地址。

① 备注

组策略、企业策略和网络上的其他限制可能会阻止这些功能成功完成。 在尝试使用这些函数之前，请确保应用程序具有必要的网络权限。

使用 AddIPAddress

1. 声明名为 `NTEContext` 和 `NTEInstance` 的 `ULONG` 变量，两者都初始化为零。

① 备注

变量 `NTEContext` 是 [DeleteIPAddress](#) 函数的唯一参数；若要删除添加的 IP 地址，`NTEContext` 必须存储且保持不变。

C++

```
ULONG NTEContext = 0;  
ULONG NTEInstance = 0;
```

① 备注

2. 分别声明名为 iaIPAddress 和 iaIPMask 的结构的变量。这些值只是无符号整数。iaIPAddress 使用 inet_addr 函数初始化和 iaIPMask 变量。

C++

```
UINT iaIPAddress;
UINT iaIPMask;

iaIPAddress = inet_addr("192.168.0.5");
iaIPMask     = inet_addr("255.255.255.0");
```

3. 调用 AddIPAddress 函数以添加 IPv4 地址。检查错误并将错误值返回到 DWORD 变量 dwRetVal (进行更广泛的错误检查)。

C++

```
dwRetVal = AddIPAddress(iaIPAddress, iaIPMask, pIPAddrTable-
    >table[0].dwIndex,
                           &NTEContext, &NTEInstance);
if (dwRetVal != NO_ERROR) {
    printf("AddIPAddress call failed with %d\n", dwRetVal);
}
```

① 备注

第三个参数是适配器索引，可以通过调用 GetIpAddrTable 函数来获取该索引。假定此函数返回的变量名为 pIPAddrTable。有关 GetIpAddrTable 函数的帮助，请参阅 使用 GetIpAddrTable 管理 IP 地址。

使用 DeleteIPAddress

- 调用 DeleteIPAddress 函数，将 NTEContext 变量作为其参数传递。检查错误并将错误值返回到 DWORD 变量 dwRetVal (进行更广泛的错误检查)。

C++

```
dwRetVal = DeleteIPAddress(NTEContext);
if (dwRetVal != NO_ERROR) {
    printf("\tDeleteIPAddress failed with error: %d\n", dwRetVal);
}
```

① 备注

若要使用 `DeleteIPAddress`，必须先调用 `AddIPAddress` 以获取句柄 `NTEContext`。前面的过程假定 `AddIPAddress` 已在代码中的某个位置调用，并且 `NTEContext` 已保存且未损坏。

下一步： [使用 `GetIpStatistics` 检索信息](#)

上一步： [使用 `IpReleaseAddress` 和 `IpRenewAddress` 管理 DHCP 租约](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

使用 GetIpStatistics 检索信息

项目 • 2023/06/13

[GetIpStatistics](#) 函数使用与系统关联的当前 IP 统计信息的信息填充指向[MIB_IPSTATS](#)结构的指针。

使用 GetIpStatistics

1. 声明一些所需的变量。

声明将用于错误检查函数调用的 `DWORD` 变量 `dwRetVal`。 声明指向名为 `pStats` 的 [MIB_IPSTATS](#) 变量的指针，并为结构分配内存。 检查是否可以分配内存。

C++

```
MIB_IPSTATS *pStats;
DWORD dwRetVal = 0;

pStats = (MIB_IPSTATS*) malloc(sizeof(MIB_IPSTATS));

if (pStats == NULL) {
    printf("Unable to allocate memory for MIB_IPSTATS\n");
}
```

2. 使用 `pStats` 参数调用 `GetIpStatistics` 函数以检索本地计算机的 IP 统计信息。 检查错误并返回 `DWORD` 变量 `dwRetVal` 中的错误值。 如果发生错误，可以使用变量 `dwRetVal` 进行更广泛的错误检查和报告。

C++

```
dwRetVal = GetIpStatistics(pStats);
if (dwRetVal != NO_ERROR) {
    printf("GetIpStatistics call failed with %d\n", dwRetVal);
}
```

3. 如果对 `GetIpStatistics` 的调用成功，请输出 `pStats` 参数指向的 [MIB_IPSTATS](#) 结构中的一些数据。

C++

```
printf("Number of interfaces: %d\n", pStats->dwNumIf);
printf("Number of IP addresses: %d\n", pStats->dwNumAddr);
printf("Number of received datagrams: %d\n", pStats->dwInReceives);
printf("Number of outgoing datagrams requested to transmit: %d\n",
pStats->dwOutRequests);
```

4. 释放为 *pStats* 参数指向的**MIB_IPSTATS**结构分配的内存。在应用程序不再需要 *pStats* 参数返回的数据后，应执行此操作。

C++

```
if (pStats)
    free(pStats);
```

[下一步：使用 GetTcpStatistics 检索信息](#)

[上一步：使用 AddIPAddress 和 DeleteIPAddress 管理 IP 地址](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

使用 GetTcpStatistics 检索信息

项目 • 2023/06/13

[GetTcpStatistics](#) 函数使用有关本地计算机的 TCP 协议统计信息的信息填充指向 [MIB_TCPSTATS](#) 结构的指针。

使用 GetTcpStatistics

1. 声明一些所需的变量。

声明将用于错误检查函数调用的 DWORD 变量 `dwRetVal`。 声明指向名为 `pTCPStats` 的 [MIB_TCPSTATS](#) 变量的指针，并为结构分配内存。 检查是否可以分配内存。

C++

```
DWORD dwRetVal = 0;
PMIB_TCPSTATS pTCPStats;

pTCPStats = (MIB_TCPSTATS *) malloc(sizeof(MIB_TCPSTATS));
if (pTCPStats == NULL) {
    printf("Error allocating memory\n");
}
```

2. 使用 `pTCPStats` 参数调用 `GetTcpStatistics` 函数，以检索本地计算机上 IPv4 的 TCP 统计信息。 检查错误并返回 DWORD 变量 `dwRetVal` 中的错误值。 如果发生错误，`dwRetVal` 可以使用变量进行更广泛的错误检查和报告。

C++

```
if ((dwRetVal = GetTcpStatistics(pTCPStats)) != NO_ERROR) {
    printf("GetTcpStatistics failed with error: %ld\n", dwRetVal);
}
```

3. 如果调用成功，则访问 `pTCPStats` 参数指向 [MIB_TCPSTATS](#) 返回的数据。

C++

```
printf("\tNumber of active opens: %u\n", pTCPStats->dwActiveOpens);
printf("\tNumber of passive opens: %u\n", pTCPStats->dwPassiveOpens);
printf("\tNumber of segments received: %u\n", pTCPStats->dwInSegs);
printf("\tNumber of segments transmitted: %u\n", pTCPStats->dwOutSegs);
printf("\tNumber of total connections: %u\n", pTCPStats->dwNumConns);
```

4. 释放为 *pTCPStats* 参数指向的 MIB_TCPSTATS 结构分配的内存。在应用程序不再需要 *pTCPStats* 参数返回的数据后，应执行此操作。

C++

```
if (pTCPStats)
    free(pTCPStats);
```

下一步： [使用 GetIpStatistics 检索信息](#)

上一步： [使用 GetIpStatistics 检索信息](#)

完整的源代码

- [完整的 IP 帮助程序应用程序源代码](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

完整的 IP 帮助程序应用程序源代码

项目 • 2024/01/28

下面是 IP 帮助程序应用程序的完整源代码。已将其他错误检查添加到源代码中。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>

#include <winsock2.h>
#include <ws2tcpip.h>

#include <iphlpapi.h>

#include <stdio.h>
#include <time.h>

// Need to link with Iphlpapi.lib and Ws2_32.lib
#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))
/* Note: could also use malloc() and free() */

int main()
{
    /* Some general variables */
    ULONG ulOutBufLen;
    DWORD dwRetVal;
    unsigned int i;

    /* variables used for GetNetworkParams */
    FIXED_INFO *pFixedInfo;
    IP_ADDR_STRING *pIPAddr;

    /* variables used for GetAdapterInfo */
    IP_ADAPTER_INFO *pAdapterInfo;
    IP_ADAPTER_INFO *pAdapter;

    /* variables used to print DHCP time info */
    struct tm newtime;
    char buffer[32];
    errno_t error;

    /* variables used for GetInterfaceInfo */
    IP_INTERFACE_INFO *pInterfaceInfo;
```

```

/* variables used for GetIpAddrTable */
MIB_IPADDRTABLE *pIPAddrTable;
DWORD dwSize;
IN_ADDR IPAddr;
char *strIPAddr;

/* variables used for AddIpAddress */
UINT iaIPAddress;
UINT imIPMask;
ULONG NTEContext;
ULONG NTEInstance;

/* variables used for GetIpStatistics */
MIB_IPSTATS *pStats;

/* variables used for GetTcpStatistics */
MIB_TCPSTATS *pTCPStats;

printf("-----\n");
printf("This is GetNetworkParams\n");
printf("-----\n");

pFixedInfo = (FIXED_INFO *) MALLOC(sizeof(FIXED_INFO));
if (pFixedInfo == NULL) {
    printf("Error allocating memory needed to call GetNetworkParams\n");
    return 1;
}
ulOutBufLen = sizeof(FIXED_INFO);

if (GetNetworkParams(pFixedInfo, &ulOutBufLen) == ERROR_BUFFER_OVERFLOW)
{
    FREE(pFixedInfo);
    pFixedInfo = (FIXED_INFO *) MALLOC(ulOutBufLen);
    if (pFixedInfo == NULL) {
        printf("Error allocating memory needed to call
GetNetworkParams\n");
        return 1;
    }
}

if (dwRetVal = GetNetworkParams(pFixedInfo, &ulOutBufLen) != NO_ERROR) {
    printf("GetNetworkParams failed with error %d\n", dwRetVal);
    if (pFixedInfo)
        FREE(pFixedInfo);
    return 1;
} else {
    printf("\tHost Name: %s\n", pFixedInfo->HostName);
    printf("\tDomain Name: %s\n", pFixedInfo->DomainName);
    printf("\tDNS Servers:\n");
    printf("\t\t%s\n", pFixedInfo->DnsServerList.IpAddress.String);

    pIPAddr = pFixedInfo->DnsServerList.Next;
    while (pIPAddr) {
        printf("\t\t%s\n", pIPAddr->IpAddress.String);

```

```

        pIPAddr = pIPAddr->Next;
    }

    printf("\tNode Type: ");
    switch (pFixedInfo->NodeType) {
    case 1:
        printf("%s\n", "Broadcast");
        break;
    case 2:
        printf("%s\n", "Peer to peer");
        break;
    case 4:
        printf("%s\n", "Mixed");
        break;
    case 8:
        printf("%s\n", "Hybrid");
        break;
    default:
        printf("\n");
    }

    printf("\tNetBIOS Scope ID: %s\n", pFixedInfo->ScopeId);

    if (pFixedInfo->EnableRouting)
        printf("\tIP Routing Enabled: Yes\n");
    else
        printf("\tIP Routing Enabled: No\n");

    if (pFixedInfo->EnableProxy)
        printf("\tWINS Proxy Enabled: Yes\n");
    else
        printf("\tWINS Proxy Enabled: No\n");

    if (pFixedInfo->EnableDns)
        printf("\tNetBIOS Resolution Uses DNS: Yes\n");
    else
        printf("\tNetBIOS Resolution Uses DNS: No\n");
}

/* Free allocated memory no longer needed */
if (pFixedInfo) {
    FREE(pFixedInfo);
    pFixedInfo = NULL;
}

printf("-----\n");
printf("This is GetAdaptersInfo\n");
printf("-----\n");

pAdapterInfo = (IP_ADAPTER_INFO *) MALLOC(sizeof(IP_ADAPTER_INFO));
if (pAdapterInfo == NULL) {
    printf("Error allocating memory needed to call GetAdapterInfo\n");
    return 1;
}
ulOutBufLen = sizeof(IP_ADAPTER_INFO);

```

```

    if (GetAdaptersInfo(pAdapterInfo, &ulOutBufLen) ==
ERROR_BUFFER_OVERFLOW) {
        FREE(pAdapterInfo);
        pAdapterInfo = (IP_ADAPTER_INFO *) MALLOC(ulOutBufLen);
        if (pAdapterInfo == NULL) {
            printf("Error allocating memory needed to call
GetAdapterInfo\n");
            return 1;
        }
    }

    if ((dwRetVal = GetAdaptersInfo(pAdapterInfo, &ulOutBufLen)) !=
NO_ERROR) {
        printf("GetAdaptersInfo failed with error %d\n", dwRetVal);
        if (pAdapterInfo)
            FREE(pAdapterInfo);
        return 1;
    }

pAdapter = pAdapterInfo;
while (pAdapter) {
    printf("\tAdapter Name: \t%s\n", pAdapter->AdapterName);
    printf("\tAdapter Desc: \t%s\n", pAdapter->Description);
    printf("\tAdapter Addr: \t");
    for (i = 0; i < (int) pAdapter->AddressLength; i++) {
        if (i == (pAdapter->AddressLength - 1))
            printf("%.2X\n", (int) pAdapter->Address[i]);
        else
            printf("%.2X-", (int) pAdapter->Address[i]);
    }
    printf("\tIP Address: \t%s\n",
           pAdapter->IpAddressList.IpAddress.String);
    printf("\tIP Mask: \t%s\n", pAdapter->IpAddressList.IpMask.String);

    printf("\tGateway: \t%s\n", pAdapter->GatewayList.IpAddress.String);
    printf("\t***\n");

    if (pAdapter->DhcpEnabled) {
        printf("\tDHCP Enabled: \tYes\n");
        printf("\tDHCP Server: \t%s\n",
               pAdapter->DhcpServer.IpAddress.String);

        printf("\tLease Obtained: ");
        /* Display local time */
        error = _localtime32_s(&newtime, (_time32_t*) &pAdapter-
>LeaseObtained);
        if (error)
            printf("\tInvalid Argument to _localtime32_s\n");

        else {
            // Convert to an ASCII representation
            error = asctime_s(buffer, 32, &newtime);
            if (error)
                printf("Invalid Argument to asctime_s\n");
        }
    }
}

```

```

        else
            /* asctime_s returns the string terminated by \n\0 */
            printf("%s", buffer);
    }

    printf("\tLease Expires: ");
    error = _localtime32_s(&newtime, (_time32_t*) &pAdapter-
>LeaseExpires);
    if (error)
        printf("Invalid Argument to _localtime32_s\n");
    else {
        // Convert to an ASCII representation
        error = asctime_s(buffer, 32, &newtime);
        if (error)
            printf("Invalid Argument to asctime_s\n");
        else
            /* asctime_s returns the string terminated by \n\0 */
            printf("%s", buffer);
    }
} else
    printf("\tDHCP Enabled: \tNo\n");

if (pAdapter->HaveWins) {
    printf("\tHave Wins: \tYes\n");
    printf("\tPrimary Wins Server: \t%s\n",
           pAdapter->PrimaryWinsServer.IpAddress.String);
    printf("\tSecondary Wins Server: \t%s\n",
           pAdapter->SecondaryWinsServer.IpAddress.String);
} else
    printf("\tHave Wins: \tNo\n");

printf("\n");
pAdapter = pAdapter->Next;
}

printf("-----\n");
printf("This is GetInterfaceInfo\n");
printf("-----\n");

pInterfaceInfo = (IP_INTERFACE_INFO *) MALLOC(sizeof
(IP_INTERFACE_INFO));
if (pInterfaceInfo == NULL) {
    printf("Error allocating memory needed to call GetInterfaceInfo\n");
    return 1;
}
ulOutBufLen = sizeof (IP_INTERFACE_INFO);
if (GetInterfaceInfo(pInterfaceInfo, &ulOutBufLen) ==
    ERROR_INSUFFICIENT_BUFFER) {
    FREE(pInterfaceInfo);
    pInterfaceInfo = (IP_INTERFACE_INFO *) MALLOC(ulOutBufLen);
    if (pInterfaceInfo == NULL) {
        printf("Error allocating memory needed to call
GetInterfaceInfo\n");
        return 1;
    }
}

```

```

        printf("\t The size needed for the output buffer ulLen = %ld\n",
               ulOutBufLen);
    }

    if ((dwRetVal = GetInterfaceInfo(pInterfaceInfo, &ulOutBufLen)) ==
NO_ERROR) {
        printf("\tNum Adapters: %ld\n\n", pInterfaceInfo->NumAdapters);
        for (i = 0; i < (unsigned int) pInterfaceInfo->NumAdapters; i++) {
            printf("\tAdapter Index[%d]: %ld\n", i,
                   pInterfaceInfo->Adapter[i].Index);
            printf("\tAdapter Name[%d]: %ws\n\n", i,
                   pInterfaceInfo->Adapter[i].Name);
        }
        printf("GetInterfaceInfo call succeeded.\n");
    } else {
        LPVOID lpMsgBuf = NULL;

        if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL, dwRetVal,
MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),           // Default language
(LPTSTR) & lpMsgBuf, 0, NULL)) {
            printf("\tError: %s", lpMsgBuf);
        }
        LocalFree(lpMsgBuf);
    }

/* If DHCP enabled, release and renew the IP address */
/* THIS WORKS BUT IT TAKES A LONG TIME AND INTERRUPTS NET CONNECTIONS */
if (pAdapterInfo->DhcpEnabled && pInterfaceInfo->NumAdapters) {
    printf("Calling IpReleaseAddress for Adapter[%d]\n", 0);
    if ((dwRetVal =
         IpReleaseAddress(&pInterfaceInfo->Adapter[0])) == NO_ERROR) {
        printf("Ip Release succeeded.\n");
    }
    if ((dwRetVal =
         IpRenewAddress(&pInterfaceInfo->Adapter[0])) == NO_ERROR) {
        printf("Ip Renew succeeded.\n");
    }
}

/* Free allocated memory no longer needed */
if (pAdapterInfo) {
    FREE(pAdapterInfo);
    pAdapterInfo = NULL;
}
if (pInterfaceInfo) {
    FREE(pInterfaceInfo);
    pInterfaceInfo = NULL;
}

printf("-----\n");
printf("This is GetIpAddrTable\n");
printf("-----\n");

pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(sizeof (MIB_IPADDRTABLE));

```

```

if (pIPAddrTable == NULL) {
    printf("Error allocating memory needed to call GetIpAddrTable\n");
    return 1;
}
dwSize = 0;
IPAddr.S_un.S_addr = ntohl(pIPAddrTable->table[1].dwAddr);
strIPAddr = inet_ntoa(IPAddr);

if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==
ERROR_INSUFFICIENT_BUFFER) {
    FREE(pIPAddrTable);
    pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(dwSize);
    if (pIPAddrTable == NULL) {
        printf("Error allocating memory needed to call
GetIpAddrTable\n");
        return 1;
    }
}

if ((dwRetVal = GetIpAddrTable(pIPAddrTable, &dwSize, 0)) != NO_ERROR) {
    printf("GetIpAddrTable failed with error %d\n", dwRetVal);
    if (pIPAddrTable)
        FREE(pIPAddrTable);
    return 1;
}

printf("\tNum Entries: %ld\n", pIPAddrTable->dwNumEntries);
for (i = 0; i < (unsigned int) pIPAddrTable->dwNumEntries; i++) {
    printf("\n\tInterface Index[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwIndex);
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwAddr;
    printf("\tIP Address[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwMask;
    printf("\tSubnet Mask[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwBcastAddr;
    printf("\tBroadCast[%d]:\t%s (%ld)\n", i, inet_ntoa(IPAddr),
           pIPAddrTable->table[i].dwBcastAddr);
    printf("\tReassembly size[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwReasmSize);
    printf("\tAddress Index[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwIndex);
    printf("\tType and State[%d]:", i);
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_PRIMARY)
        printf("\tPrimary IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DYNAMIC)
        printf("\tDynamic IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DISCONNECTED)
        printf("\tAddress is on disconnected interface");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DELETED)
        printf("\tAddress is being deleted");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_TRANSIENT)
        printf("\tTransient address");
    printf("\n");
}

```

```

iaIPAddress = inet_addr("192.168.0.27");
imIPMask = inet_addr("255.255.255.0");

NTEContext = 0;
NTEInstance = 0;

if ((dwRetVal = AddIPAddress(iaIPAddress,
                             imIPMask,
                             pIPAddrTable->table[0].dwIndex,
                             &NTEContext, &NTEInstance)) != NO_ERROR) {

    LPVOID lpMsgBuf;
    printf("\tError adding IP address.\n");

    if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL, dwRetVal,
MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),           // Default language
(LPTSTR) & lpMsgBuf, 0, NULL)) {
        printf("\tError: %s", lpMsgBuf);
    }
    LocalFree(lpMsgBuf);
}

if ((dwRetVal = DeleteIPAddress(NTEContext)) != NO_ERROR) {
    printf("DeleteIPAddress failed with error %d\n", dwRetVal);
}

/* Free allocated memory no longer needed */
if (pIPAddrTable) {
    FREE(pIPAddrTable);
    pIPAddrTable = NULL;
}

printf("-----\n");
printf("This is GetIPStatistics()\n");
printf("-----\n");

pStats = (MIB_IPSTATS *) MALLOC(sizeof(MIB_IPSTATS));
if (pStats == NULL) {
    printf("Error allocating memory needed to call GetIpStatistics\n");
    return 1;
}

if ((dwRetVal = GetIpStatistics(pStats)) != NO_ERROR) {
    printf("GetIPStatistics failed with error %d\n", dwRetVal);
    if (pStats)
        FREE(pStats);
    return 1;
}

printf("\tNumber of IP addresses: %ld\n", pStats->dwNumAddr);
printf("\tNumber of Interfaces: %ld\n", pStats->dwNumIf);
printf("\tReceives: %ld\n", pStats->dwInReceives);
printf("\tOut Requests: %ld\n", pStats->dwOutRequests);

```

```

printf("\tRoutes: %ld\n", pStats->dwNumRoutes);
printf("\tTimeout Time: %ld\n", pStats->dwReasmTimeout);
printf("\tIn Delivers: %ld\n", pStats->dwInDelivers);
printf("\tIn Discards: %ld\n", pStats->dwInDiscards);
printf("\tTotal In: %ld\n", pStats->dwInDelivers + pStats-
>dwInDiscards);
printf("\tIn Header Errors: %ld\n", pStats->dwInHdrErrors);

/* Free allocated memory no longer needed */
if (pStats) {
    FREE(pStats);
    pStats = NULL;
}

printf("-----\n");
printf("This is GetTCPStatistics()\n");
printf("-----\n");

pTCPStats = (MIB_TCPSTATS *) MALLOC(sizeof (MIB_TCPSTATS));
if (pTCPStats == NULL) {
    printf("Error allocating memory needed to call GetTcpStatistics\n");
    return 1;
}

if ((dwRetVal = GetTcpStatistics(pTCPStats)) != NO_ERROR) {
    printf("GetTcpStatistics failed with error %d\n", dwRetVal);
    if (pTCPStats)
        FREE(pTCPStats);
    return 1;
}

printf("\tActive Opens: %ld\n", pTCPStats->dwActiveOpens);
printf("\tPassive Opens: %ld\n", pTCPStats->dwPassiveOpens);
printf("\tSegments Recv: %ld\n", pTCPStats->dwInSegs);
printf("\tSegments Xmit: %ld\n", pTCPStats->dwOutSegs);
printf("\tTotal # Conxs: %ld\n", pTCPStats->dwNumConns);

/* Free allocated memory no longer needed */
if (pTCPStats) {
    FREE(pTCPStats);
    pTCPStats = NULL;
}

return 0;
}

```

此页面是否有帮助?

是

否

包时间戳

项目 • 2023/06/12

简介

每当收到或传输数据包时，许多网络接口卡 (NIC 或网络适配器) 都可以在其硬件中生成时间戳。时间戳是使用 NIC 自己的硬件时钟生成的。此功能特别由精确时间协议 (PTP) 使用，这是一种时间同步协议。PTP 预配以在协议本身中使用此类硬件时间戳。

例如，时间戳可用于计算计算机网络堆栈中的数据包在发送到线路或从线路接收数据包之前所花费的时间。然后，PTP 可以使用这些计算来提高时间同步的准确性。网络适配器的数据包时间戳支持有时专门用于 PTP 协议。在其他情况下，会提供更常规的支持。

时间戳 API 使 Windows 能够支持 PTP 版本 2 协议的网络适配器的硬件时间戳功能。总的来说，这些功能包括为网络适配器的驱动程序提供支持时间戳的功能，以及用户模式应用程序通过 [Windows 套接字](#) 使用与数据包关联的时间戳 (请参阅 [Winsock 时间戳](#))。此外，还可以生成软件时间戳，这允许网络驱动程序在软件中生成时间戳。此类软件时间戳由 NIC 驱动程序使用 [与 QueryPerformanceCounter \(QPC\) 等效的内核模式生成](#)。但是，不支持同时 [启用硬件和软件时间戳](#)。

具体而言，本主题中所述的 IP 协议帮助程序 (IP 帮助程序) 数据包时间戳 API，使用户模式应用程序能够确定网络适配器的时间戳功能，以及以交叉时间戳的形式从网络适配器查询时间戳，(如下) 所述。

支持精度时间协议版本 2

如前所述，Windows 中时间戳支持的主要目标是支持精度时间协议版本 2 (PTPv2) 协议。在 PTPv2 中，并非所有消息都需要时间戳。具体而言，PTP 事件消息确实使用时间戳。目前，支持范围限定为基于用户数据报协议的 PTPv2 (UDP)。不支持通过原始以太网进行 PTP。

在 2 步模式下运行的 PTPv2 支持时间戳。2 步骤是指 PTP 数据包中的实际时间戳不是在硬件中动态生成，而是从硬件中检索并作为单独的消息 (传递的模式，例如，使用后续消息)。

总之，可以在 PTPv2 应用程序中使用 Internet 协议帮助程序 (IP 帮助程序) 数据包时间戳 API 以及 Winsock 的时间戳支持，以提高其时间同步准确性。

检索网络适配器的时间戳功能

PTP 时间同步服务等应用程序需要确定网络适配器的时间戳功能。然后，使用检索到的功能，应用程序可以决定是否要使用时间戳。

即使网络适配器 确实 支持时间戳，也要求默认关闭此功能。适配器在指示时打开时间戳。Windows 为应用程序提供 API 来检索硬件的功能，以及打开的功能。

若要检索网络适配器支持的时间戳功能，请调用

[GetInterfaceSupportedTimestampCapabilities](#) 函数，(网络适配器的 LUID) 提供本地唯一标识符，并返回以 [INTERFACE_TIMESTAMP_CAPABILITIES](#) 对象的形式检索支持的时间戳功能。

从 [GetInterfaceSupportedTimestampCapabilities](#) 返回的代码指示调用是否成功，以及是否检索了填充 [INTERFACE_TIMESTAMP_CAPABILITIES](#) 值。

若要检索网络适配器当前启用的时间戳功能，请调用

[GetInterfaceActiveTimestampCapabilities](#) 函数，(网络适配器的 LUID) 提供本地唯一标识符，并返回以 [INTERFACE_TIMESTAMP_CAPABILITIES](#) 对象的形式检索已启用的时间戳功能。

同样，从 [GetInterfaceActiveTimestampCapabilities](#) 返回的代码指示成功还是失败，以及是否检索了有效的 [INTERFACE_TIMESTAMP_CAPABILITIES](#) 值。

网络适配器可以支持各种时间戳功能。例如，某些适配器可以在发送和接收期间对每个数据包进行时间戳，而其他适配器仅支持 PTPv2 数据包。

[INTERFACE_TIMESTAMP_CAPABILITIES](#) 结构描述了网络适配器支持的确切功能。

从网络适配器检索交叉时间戳

使用硬件时间戳时，PTP 应用程序需要 (建立关系，例如，通过在网络适配器的硬件时钟与系统时钟之间) 适当的数学技术。这是必需的，以便可以将表示一个时钟单位中的时间的值转换为另一个时钟单位。为此，提供了交叉时间戳，应用程序可以定期对交叉时间戳进行采样，以建立这种关系。

为此，请调用 [CaptureInterfaceHardwareCrossTimestamp](#) 函数，(网络适配器的 LUID) 提供本地唯一标识符，并返回以 [INTERFACE_HARDWARE_CROSSTIMESTAMP](#) 对象的形式从网络适配器检索时间戳。

时间戳功能更改通知

若要在网络适配器的时间戳功能发生更改时收到通知，请调用

[RegisterInterfaceTimestampConfigChange](#) 函数，提供指向已实现的回调函数的指针，以及调用方分配的可选上下文。

[RegisterInterfaceTimestampConfigChange](#) 返回一个句柄，随后可以传递给 [UnregisterInterfaceTimestampConfigChange](#)，以便取消注册回调函数。

代码示例 1 - 检索时间戳功能和交叉时间戳

C

```
// main.cpp in a Console App project.

#include <stdio.h>
#include <winsock2.h>
#include <iphlpapi.h>
#pragma comment(lib, "Iphlpapi")

BOOL
IsPTPv2HardwareTimestampingSupportedForIPv4(PINTERFACE_TIMESTAMP_CAPABILITIES timestampCapabilities)
{
    // Supported if both receive and transmit side support is present
    if (((timestampCapabilities->HardwareCapabilities.PtpV2OverUdpIPv4EventMessageReceive) ||
        (timestampCapabilities->HardwareCapabilities.PtpV2OverUdpIPv4AllMessageReceive) ||
        (timestampCapabilities->HardwareCapabilities.AllReceive)) &&
        ((timestampCapabilities->HardwareCapabilities.PtpV2OverUdpIPv4EventMessageTransmit) ||
        (timestampCapabilities->HardwareCapabilities.PtpV2OverUdpIPv4AllMessageTransmit) ||
        (timestampCapabilities->HardwareCapabilities.TaggedTransmit) ||
        (timestampCapabilities->HardwareCapabilities.AllTransmit)))
    {
        return TRUE;
    }

    return FALSE;
}

BOOL
IsPTPv2HardwareTimestampingSupportedForIPv6(PINTERFACE_TIMESTAMP_CAPABILITIES timestampCapabilities)
{
    // Supported if both receive and transmit side support is present
    if (((timestampCapabilities->HardwareCapabilities.PtpV2OverUdpIPv6EventMessageReceive) ||
        (timestampCapabilities->HardwareCapabilities.PtpV2OverUdpIPv6AllMessageReceive) ||
        (timestampCapabilities->HardwareCapabilities.AllReceive)) &&
        ((timestampCapabilities->HardwareCapabilities.PtpV2OverUdpIPv6EventMessageTransmit) ||
        (timestampCapabilities->
```

```

>HardwareCapabilities.PtpV2OverUdpIPv6AllMessageTransmit) ||
    (timestampCapabilities->HardwareCapabilities.TaggedTransmit) ||
    (timestampCapabilities->HardwareCapabilities.AllTransmit)))
{
    return TRUE;
}

return FALSE;
}

enum SupportedTimestampType
{
    TimestampTypeNone = 0,
    TimestampTypeSoftware = 1,
    TimestampTypeHardware = 2
};

// This function checks and returns the supported timestamp capabilities for
// an interface for
// a PTPv2 application
SupportedTimestampType
CheckActiveTimestampCapabilitiesForPtpv2(NET_LUID interfaceLuid)
{
    DWORD result = NO_ERROR;
    INTERFACE_TIMESTAMP_CAPABILITIES timestampCapabilities;
    SupportedTimestampType supportedType = TimestampTypeNone;

    result = GetInterfaceActiveTimestampCapabilities(
        &interfaceLuid,
        &timestampCapabilities);
    if (result != NO_ERROR)
    {
        printf("Error retrieving hardware timestamp capabilities: %d\n",
result);
        goto Exit;
    }

    if (IsPTPv2HardwareTimestampingSupportedForIPv4(&timestampCapabilities) &&
        IsPTPv2HardwareTimestampingSupportedForIPv6(&timestampCapabilities))
    {
        supportedType = TimestampTypeHardware;
        goto Exit;
    }
    else
    {
        if (((timestampCapabilities.SoftwareCapabilities.AllReceive) &&
            ((timestampCapabilities.SoftwareCapabilities.AllTransmit) ||
            (timestampCapabilities.SoftwareCapabilities.TaggedTransmit)))
        {
            supportedType = TimestampTypeSoftware;
        }
    }
}

Exit:

```

```

        return supportedType;
    }

    // Helper function which does the correlation between hardware and system
    // clock
    // using mathematical techniques
    void
    ComputeCorrelationOfHardwareAndSystemTimestamps(INTERFACE_HARDWARE_CROSSTIME
STAMP *crossTimestamp);

    // An application would call this function periodically to gather a set
    // of matching timestamps for use in converting hardware timestamps to
    // system timestamps
    DWORD
    RetrieveAndProcessCrossTimestamp(NET_LUID interfaceLuid)
{
    DWORD result = NO_ERROR;
    INTERFACE_HARDWARE_CROSSTIMESTAMP crossTimestamp;

    result = CaptureInterfaceHardwareCrossTimestamp(
        &interfaceLuid,
        &crossTimestamp);
    if (result != NO_ERROR)
    {
        printf("Error retrieving cross timestamp for the interface: %d\n",
result);
        goto Exit;
    }

    // Process crossTimestamp further to create a relation between the
    // hardware clock
    // of the NIC and the QPC values using appropriate mathematical
    // techniques
    ComputeCorrelationOfHardwareAndSystemTimestamps(&crossTimestamp);

Exit:
    return result;
}

int main()
{
}

```

代码示例 2 - 注册时间戳功能更改通知

此示例演示应用程序如何使用端到端时间戳。

C

```

// main.cpp in a Console App project.

#include <stdlib.h>
```

```

#include <stdio.h>
#include <winsock2.h>
#include <mswsock.h>
#include <iphlpapi.h>
#include <mstcpip.h>
#pragma comment(lib, "Ws2_32")
#pragma comment(lib, "Iphlpapi")

// Globals and function declarations used by the application.
// The sample functions and skeletons demonstrate:
// - Checking timestamp configuration for an interface to determine if
timestamping can be used
// - If timestamping is enabled, starts tracking changes in timestamp
configuration
// - Performing correlation between hardware and system timestamps using
cross timestamps
// on a separate thread depending on the timestamp type configured
// - Receiving a packet and computing the latency between when the timestamp
// was generated on packet reception, and when the packet was received by
// the application through the socket
// The sample tries to demonstrate how an application could use timestamps.
It is not thread safe
// and does not do exhaustive error checking.
// Lot of the functions are provided as skeletons, or only declared and
invoked
// but are not defined. It is up to
// the application to implement these suitably.

// An application could use the functions below by e.g.
// - Call InitializeTimestampingForInterface for the interface it wants to
track for timestamping capability.
// - Call EstimateReceiveLatency to estimate the receive latency of a packet
depending on the timestamp
// type configured for the interface.

enum SupportedTimestampType
{
    TimestampTypeNone = 0,
    TimestampTypeSoftware = 1,
    TimestampTypeHardware = 2
};

// interfaceBeingTracked is the interface the PTPv2 application
// intends to use for timestamping purpose.
wchar_t* interfaceBeingTracked;

// The active timestamping type determined for
// interfaceBeingTracked.
SupportedTimestampType timestampTypeEnabledForInterface;

HANDLE correlationThread;
HANDLE threadStopEvent;
HIFTIMESTAMPCHANGE TimestampChangeNotificationHandle = NULL;

// Function from sample above to check if an interface supports timestamping

```

```

for PTPv2.

SupportedTimestampType CheckActiveTimestampCapabilitiesForPtpv2(NET_LUID
interfaceLuid);

// Function from sample above to retrieve cross timestamps and process them
// further.
DWORD RetrieveAndProcessCrossTimestamp(NET_LUID interfaceLuid);

// Helper function which registers for timestamp configuration changes.
DWORD RegisterTimestampChangeNotifications();

// Callback function which is invoked when timestamp configuration changes
// for some network interface.
INTERFACE_TIMESTAMP_CONFIG_CHANGE_CALLBACK TimestampConfigChangeCallback;

// Function which does the correlation between hardware and system clock
// using mathematical techniques. It is periodically invoked and provided
// a sample of cross timestamp to compute a correlation.
void
ComputeCorrelationOfHardwareAndSystemTimestamps(INTERFACE_HARDWARE_CROSSTIME
STAMP *crossTimestamp);

// Helper function which converts a hardware timestamp from the NIC clock
// to system timestamp (QPC) values. It is assumed that this works together
// with the ComputeCorrelationOfHardwareAndSystemTimestamps function
// to derive the correlation.
ULONG64 ConvertHardwareTimestampToQpc(ULONG64 HardwareTimestamp);

// Start function of thread which periodically samples
// cross timestamps to correlate hardware and software timestamps.
DWORD WINAPI CorrelateHardwareAndSystemTimestamps(LPVOID);

// Helper function which starts a new thread at
CorrelateHardwareAndSystemTimestamps.
DWORD StartCorrelatingHardwareAndSystemTimestamps();

// Helper function which restarts correlation when some change is detected.
DWORD RestartCorrelatingHardwareAndSystemTimestamps();

// Stops the correlation thread.
DWORD StopCorrelatingHardwareAndSystemTimestamps();

DWORD
FindInterfaceFromFriendlyName(wchar_t* friendlyName, NET_LUID*
interfaceLuid)
{
    DWORD result = 0;
    ULONG flags = 0;
    ULONG outBufLen = 0;
    PIP_ADAPTER_ADDRESSES pAddresses = NULL;
    PIP_ADAPTER_ADDRESSES currentAddresses = NULL;

    result = GetAdaptersAddresses(0,
        flags,
        NULL,

```

```

    pAddresses,
    &outBufLen);
if (result == ERROR_BUFFER_OVERFLOW)
{
    pAddresses = (PIP_ADAPTER_ADDRESSES)malloc(outBufLen);

    result = GetAdaptersAddresses(0,
        flags,
        NULL,
        pAddresses,
        &outBufLen);
    if (result != NO_ERROR)
    {
        goto Done;
    }
}
else if (result != NO_ERROR)
{
    goto Done;
}

currentAddresses = pAddresses;
while (currentAddresses != NULL)
{
    if (wcscmp(friendlyName, currentAddresses->FriendlyName) == 0)
    {
        result = ConvertInterfaceIndexToLuid(currentAddresses->IfIndex,
interfaceLuid);
        goto Done;
    }

    currentAddresses = currentAddresses->Next;
}

result = ERROR_NOT_FOUND;

```

Done:

```

if (pAddresses != NULL)
{
    free(pAddresses);
}

return result;
}

// This function checks if an interface is suitable for
// timestamping for PTPv2. If so, it registers for timestamp
// configuration changes and initializes some globals.
// If hardware timestamping is enabled it also starts
// correlation thread.

DWORD
InitializeTimestampingForInterface(wchar_t* friendlyName)
{
    DWORD error;

```

```
SupportedTimestampType supportedType = TimestampTypeNone;

NET_LUID interfaceLuid;

error = FindInterfaceFromFriendlyName(friendlyName, &interfaceLuid);
if (error != 0)
{
    return error;
}

supportedType = CheckActiveTimestampCapabilitiesForPtpv2(interfaceLuid);

if (supportedType != TimestampTypeNone)
{
    error = RegisterTimestampChangeNotifications();
    if (error != NO_ERROR)
    {
        return error;
    }

    if (supportedType == TimestampTypeHardware)
    {
        threadStopEvent = CreateEvent(
            NULL,
            FALSE,
            FALSE,
            NULL
        );
        if (threadStopEvent == NULL)
        {
            return GetLastError();
        }
    }

    error = StartCorrelatingHardwareAndSystemTimestamps();
    if (error != 0)
    {
        return error;
    }
}

interfaceBeingTracked = friendlyName;
timestampTypeEnabledForInterface = supportedType;

return error;
}

return ERROR_NOT_SUPPORTED;
}

DWORD
RegisterTimestampChangeNotifications()
{
    DWORD retcode = NO_ERROR;

    // Register with NULL context
```

```

    retcode =
RegisterInterfaceTimestampConfigChange(TimestampConfigChangeCallback, NULL,
&TimestampChangeNotificationHandle);
    if (retcode != NO_ERROR)
    {
        printf("Error when calling RegisterIfTimestampConfigChange %d\n",
retcode);
    }

    return retcode;
}

// The callback invoked when change in some interface's timestamping
configuration
// happens. The callback takes appropriate action based on the new
capability of the
// interface. The callback assumes that there is only 1 NIC. If multiple
NICs are being
// tracked for timestamping then the application would need to check all of
them.

VOID
WINAPI
TimestampConfigChangeCallback(
    _In_ PVOID /*CallerContext*/
)
{
    SupportedTimestampType supportedType;

    NET_LUID interfaceLuid;
    DWORD error;

    error = FindInterfaceFromFriendlyName(interfaceBeingTracked,
&interfaceLuid);
    if (error != NO_ERROR)
    {
        if (timestampTypeEnabledForInterface == TimestampTypeHardware)
        {
            StopCorrelatingHardwareAndSystemTimestamps();
            timestampTypeEnabledForInterface = TimestampTypeNone;
        }
        return;
    }

    supportedType = CheckActiveTimestampCapabilitiesForPtpv2(interfaceLuid);

    if ((supportedType == TimestampTypeHardware) &&
        (timestampTypeEnabledForInterface == TimestampTypeHardware))
    {
        // NIC could have been restarted, restart the correlation between
hardware and
        // system timestamps.
        RestartCorrelatingHardwareAndSystemTimestamps();
    }
    else if (supportedType == TimestampTypeHardware)
    {

```

```

        // Start thread correlating hardware and software timestamps
        StartCorrelatingHardwareAndSystemTimestamps();

    }

    else if (supportedType != TimestampTypeHardware)
    {
        // Hardware timestamps are not enabled, stop correlation
        StopCorrelatingHardwareAndSystemTimestamps();
    }

    timestampTypeEnabledForInterface = supportedType;
}

DWORD
StartCorrelatingHardwareAndSystemTimestamps()
{
    // Create a new thread which starts at
    CorrelateHardwareAndSoftwareTimestamps
    correlationThread = CreateThread(
        NULL,
        0,
        CorrelateHardwareAndSystemTimestamps,
        NULL,
        0,
        NULL);

    if (correlationThread == NULL)
    {
        return GetLastError();
    }
}

// Thread which periodically invokes functions to
// sample cross timestamps and use them to compute
// correlation between hardware and system timestamps.
DWORD WINAPI
CorrelateHardwareAndSystemTimestamps(LPVOID /*lpParameter*/)
{
    DWORD error;
    NET_LUID interfaceLuid;
    DWORD result;

    result = FindInterfaceFromFriendlyName(interfaceBeingTracked,
&interfaceLuid);
    if (result != 0)
    {
        return result;
    }

    while (TRUE)
    {
        error = RetrieveAndProcessCrossTimestamp(interfaceLuid);

        // Sleep and repeat till the thread gets a signal to stop
        result = WaitForSingleObject(threadStopEvent, 5000);
        if (result != WAIT_TIMEOUT)

```

```

    {
        if (result == WAIT_OBJECT_0)
        {
            return 0;
        }
        else if (result == WAIT_FAILED)
        {
            return GetLastError();
        }

        return result;
    }
}

DWORD
StopCorrelatingHardwareAndSystemTimestamps()
{
    SetEvent(threadStopEvent);
    return 0;
}

// Function which receives a packet and estimates the latency between the
// point at which receive timestamp (of appropriate type) was generated
// and when the packet was received in the app through the socket.
// The sample assumes that there is only 1 NIC in the system. This is the
// NIC which is tracked through
// interfaceBeingTracked for correlation purpose, and through which packets
// are being
// received by the socket.
// The recvmsg parameter is of type LPFN_WSARECVMMSG and an application can
// retrieve it by issuing WSAIoctl
// with SIO_GET_EXTENSION_FUNCTION_POINTER control
// and WSAID_WSARECVMMSG. Please refer to msdn.
void EstimateReceiveLatency(SOCKET sock, LPFN_WSARECVMMSG recvmsg)
{
    DWORD numBytes;
    INT error;
    CHAR data[512];
    CHAR control[WSA_CMSG_SPACE(sizeof(UINT64))] = { 0 };
    WSABUF dataBuf;
    WSABUF controlBuf;
    WSAMSG wsaMsg;
    UINT64 socketTimestamp = 0;
    ULONG64 appLevelTimestamp;
    ULONG64 packetReceivedTimestamp;

    dataBuf.buf = data;
    dataBuf.len = sizeof(data);
    controlBuf.buf = control;
    controlBuf.len = sizeof(control);
    wsaMsg.name = NULL;
    wsaMsg.namelen = 0;
    wsaMsg.lpBuffers = &dataBuf;
    wsaMsg.dwBufferCount = 1;
}

```

```

wsaMsg.Control = controlBuf;
wsaMsg.dwFlags = 0;

// Configure rx timestamp reception.
TIMESTAMPING_CONFIG config = { 0 };
config.Flags |= TIMESTAMPING_FLAG_RX;
error =
    WSAIoctl(
        sock,
        SIO_TIMESTAMPING,
        &config,
        sizeof(config),
        NULL,
        0,
        &numBytes,
        NULL,
        NULL);
if (error == SOCKET_ERROR)
{
    printf("WSAIoctl failed %d\n", WSAGetLastError());
    return;
}

error =
    recvmsg(
        sock,
        &wsaMsg,
        &numBytes,
        NULL,
        NULL);
if (error == SOCKET_ERROR)
{
    printf("recvmsg failed %d\n", WSAGetLastError());
    return;
}

if (timestampTypeEnabledForInterface != TimestampTypeNone)
{
    // Capture system timestamp (QPC) upon message reception.
    LARGE_INTEGER t1;
    QueryPerformanceCounter(&t1);
    appLevelTimestamp = t1.QuadPart;

    printf("received packet\n");

    // Look for socket rx timestamp returned via control message.
    BOOLEAN retrievedTimestamp = FALSE;
    PCMSGHDR cmsg = WSA_CMSG_FIRSTHDR(&wsaMsg);
    while (cmsg != NULL)
    {
        if (cmsg->cmsg_level == SOL_SOCKET && cmsg->cmsg_type ==
SO_TIMESTAMP)
        {
            socketTimestamp = *(PUINT64)WSA_CMSG_DATA(cmsg);
            retrievedTimestamp = TRUE;
        }
    }
}

```

```

        break;
    }
    cmsg = WSA_CMSG_NXTHDR(&wsaMsg, cmsg);
}

if (retrievedTimestamp)
{
    // Compute socket receive path latency.
    LARGE_INTEGER clockFrequency;
    ULONG64 elapsedMicroseconds;

    if (timestampTypeEnabledForInterface == TimestampTypeHardware)
    {
        packetReceivedTimestamp =
ConvertHardwareTimestampToQpc(socketTimestamp);
    }
    else
    {
        packetReceivedTimestamp = socketTimestamp;
    }

    QueryPerformanceFrequency(&clockFrequency);

    // Compute socket receive path latency.
    elapsedMicroseconds = appLevelTimestamp -
packetReceivedTimestamp;
    elapsedMicroseconds *= 1000000;
    elapsedMicroseconds /= clockFrequency.QuadPart;
    printf("RX latency estimation: %lld microseconds\n",
           elapsedMicroseconds);
}
else
{
    printf("failed to retrieve RX timestamp\n");
}
}

int main()
{
}

```

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP Helper 参考

项目 • 2023/06/12

使用以下函数和结构检索和修改本地计算机上的传输控制协议/Internet 协议 (TCP/IP) 传输的配置设置。

- IP Helper 回调
- IP Helper 枚举
- IP 帮助程序函数
- IP Helper 结构

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

IP Helper 回调

项目 • 2023/06/12

以下回调函数可用于 IP 帮助程序。

- INTERFACE_TIMESTAMP_CONFIG_CHANGE_CALLBACK
- PNETWORK_CONNECTIVITY_HINT_CHANGE_CALLBACK

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

CaptureInterfaceHardwareCrossTimestamp 函数 (iphlpapi.h)

项目 2023/08/24

检索网络适配器的跨时间戳信息。

有关详细信息和代码示例，请参阅 [数据包时间戳](#)。

① 重要

在 Windows 10 版本 2004 (10.0;内部版本 19041) 及更早版本保留此函数供系统使用，不应从代码中调用它。在更高版本中，[支持此函数](#)。

语法

C++

```
IPHLAPI_DLL_LINKAGE DWORD CaptureInterfaceHardwareCrossTimestamp(
    const NET_LUID                *InterfaceLuid,
    PINTERFACE_HARDWARE_CROSSTIMESTAMP CrossTimestamp
);
```

parameters

InterfaceLuid

类型: `_In_ CONST NET_LUID*`

网络本地唯一标识符 (要从中检索交叉时间戳的网络适配器的 LUID)。

CrossTimestamp

类型: `_Inout_ PINTERFACE_HARDWARE_CROSSTIMESTAMP`

时间戳由网络适配器以 `INTERFACE_HARDWARE_CROSSTIMESTAMP` 对象的形式返回。

返回值

类型: `DWORD`

指示成功或失败的 DWORD 返回代码。

要求

最低受支持的客户端	Windows 11 (内部版本 10.0.22000.194)
最低受支持的服务器	Windows Server 2022
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[包时间戳](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

PNETWORK_CONNECTIVITY_HINT_CHANGE_CALLBACK回调函数 (netioapi.h)

项目2023/08/25

PNETWORK_CONNECTIVITY_HINT_CHANGE_CALLBACK类型是指向在应用程序中定义的函数的指针。每当网络聚合连接级别和成本提示发生更改时，将调用函数。

使用对 [NotifyNetworkConnectivityHintChange](#) 的调用注册回调。

语法

C++

```
PNETWORK_CONNECTIVITY_HINT_CHANGE_CALLBACK  
PnetworkConnectivityHintChangeCallback;  
  
void PnetworkConnectivityHintChangeCallback(  
    [in] PVOID CallerContext,  
    [in] NL_NETWORK_CONNECTIVITY_HINT ConnectivityHint  
)  
{...}
```

parameters

[in] CallerContext

特定于用户的调用方上下文。

[in] ConnectivityHint

类型的值 [NL_NETWORK_CONNECTIVITY_HINT](#) 表示聚合连接级别和成本提示。

返回值

无

要求

最低受支持的服务器	Windows Server 版本 2004 (10.0;内部版本 19041)
目标平台	Windows
标头	netioapi.h (包括 iphlpapi.h)

请参阅

[NotifyNetworkConnectivityHintChange](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP Helper 枚举

项目 • 2023/06/12

以下枚举与 IP 帮助程序一起使用。

- [DNS_SERVER_PROPERTY_TYPE](#)
- [IF_OPER_STATUS](#)
- [IP_DAD_STATE](#)
- [IP_PREFIX_ORIGIN](#)
- [IP_SUFFIX_ORIGIN](#)
- [NET_ADDRESS_FORMAT](#)
- [NL_NETWORK_CONNECTIVITY_LEVEL_HINT](#)
- [NL_NETWORK_CONNECTIVITY_COST_HINT](#)
- [SCOPE_LEVEL](#)
- [TCP_BOOLEAN_OPTIONAL](#)
- [TCP_ESTATS_TYPE](#)
- [TCP_SOFT_ERROR](#)
- [TCP_TABLE_CLASS](#)
- [TCPIP_OWNER_MODULE_INFO_CLASS](#)
- [UDP_TABLE_CLASS](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

DNS_SERVER_PROPERTY_TYPE 枚举 (netioapi.h)

项目2023/03/16

定义常量，这些常量指定 [DNS_SERVER_PROPERTY::Property](#) 成员中保存的属性的有效性。

语法

C++

```
typedef enum _DNS_SERVER_PROPERTY_TYPE {
    DnsServerInvalidProperty = 0,
    DnsServerDohProperty
} DNS_SERVER_PROPERTY_TYPE;
```

常量

`DnsServerInvalidProperty`

值: 0

指定 [DNS_SERVER_PROPERTY::Property](#) 成员不包含有效的 DNS 服务器属性。

`DnsServerDohProperty`

指定包含在 [DNS_SERVER_PROPERTY::Property](#) 成员中的 *DohSettings* 联合成员指向有效的 DNS-over-HTTPS 服务器属性。

要求

最低受支持的客户端

Windows 10内部版本 20348

最低受支持的服务器

Windows 10内部版本 20348

标头

netioapi.h (包括 Iphlpapi.h)

另请参阅

- DNS_SERVER_PROPERTY
-

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IF_OPER_STATUS枚举 (ifdef.h)

项目2023/08/24

IF_OPER_STATUS枚举指定接口的操作状态。

语法

C++

```
typedef enum {
    IfOperStatusUp = 1,
    IfOperStatusDown,
    IfOperStatusTesting,
    IfOperStatusUnknown,
    IfOperStatusDormant,
    IfOperStatusNotPresent,
    IfOperStatusLowerLayerDown
} IF_OPER_STATUS;
```

常量

 展开表

IfOperStatusUp

值: 1

接口已启动并正常运行。 接口能够传递数据包。

IfOperStatusDown

接口未关闭且无法运行。 接口无法传递数据包。

IfOperStatusTesting

接口正在测试中。

IfOperStatusUnknown

接口状态未知。

IfOperStatusDormant

接口不是

用于传递数据包的条件。 接口未启动，但
处于挂起状态，等待某个外部事件。 此状态标识以下情况：
接口正在等待事件将其置于 up 状态。

IfOperStatusNotPresent

此状态是对向下状态的优化
指示接口已关闭，具体是因为
某些组件（例如，硬件组件）不存在
系统。

IfOperStatusLowerLayerDown

此状态是向下状态的优化。
接口可操作，但接口下方的网络层无法正常运行。

注解

IF_OPER_STATUS 枚举在 [IP_ADAPTER_ADDRESSES](#) 结构的 OperStatus 成员中使用。

要求

[+] 展开表

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	ifdef.h

另请参阅

[IP_ADAPTER_ADDRESSES](#)

反馈

此页面是否有帮助？

是

否

NL_DAD_STATE枚举 (nldef.h)

项目2023/08/25

IP_DAD_STATE 枚举指定有关 IPv4 或 IPv6 地址的重复地址检测 (DAD) 状态的信息。

语法

C++

```
typedef enum {
    NldsInvalid,
    NldsTentative,
    NldsDuplicate,
    NldsDeprecated,
    NldsPreferred,
    IpDadStateInvalid = 0,
    IpDadStateTentative,
    IpDadStateDuplicate,
    IpDadStateDeprecated,
    IpDadStatePreferred
} NL_DAD_STATE;
```

常量

[+] 展开表

NldsInvalid

NldsTentative

NldsDuplicate

NldsDeprecated

NldsPreferred

IpDadStateInvalid

值: 0

DAD 状态无效。

IpDadStateTentative

DAD 状态为暂定状态。

`IpDadStateDuplicate`

检测到重复的 IP 地址。

`IpDadStateDeprecated`

IP 地址已弃用。

`IpDadStatePreferred`

IP 地址是首选地址。

注解

`IP_DAD_STATE` 枚举在 [IP_ADAPTER_UNICAST_ADDRESS](#) 结构的 `DadState` 成员中使用。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织方式已更改，`IP_DAD_STATE` 枚举在 *Iptypes.h* 头文件自动包含的 *Nldef.h* 头文件中定义。永远不应直接使用 *Nldef.h* 和 *Iptypes.h* 头文件。

要求

 展开表

最低受支持的客户端 Windows XP [仅限桌面应用]

最低受支持的服务器 Windows Server 2003 [仅限桌面应用]

标头 `nldef.h` (包括 Windows 8 上的 `Iphlpapi.h`、Windows Server 2008 R2、Windows 7、Windows Server 2008 Windows Vista)

请参阅

[IP_ADAPTER_UNICAST_ADDRESS](#)

反馈

此页面是否有帮助？

 是

 否

NL_PREFIX_ORIGIN枚举 (nldef.h)

项目2023/08/25

IP_PREFIX_ORIGIN枚举指定 IPv4 或 IPv6 地址前缀的来源，并与 IP_ADAPTER_UNICAST_ADDRESS 结构一起使用。

语法

C++

```
typedef enum {
    IpPrefixOriginOther = 0,
    IpPrefixOriginManual,
    IpPrefixOriginWellKnown,
    IpPrefixOriginDhcp,
    IpPrefixOriginRouterAdvertisement,
    IpPrefixOriginUnchanged = 1 << 4
} NL_PREFIX_ORIGIN;
```

常量

[] 展开表

`IpPrefixOriginOther`

值: 0

IP 前缀由此枚举中定义的源以外的其他源提供。

`IpPrefixOriginManual`

IP 地址前缀是手动指定的。

`IpPrefixOriginWellKnown`

IP 地址前缀来自已知源。

`IpPrefixOriginDhcp`

IP 地址前缀由 DHCP 设置提供。

`IpPrefixOriginRouterAdvertisement`

IP 地址前缀是通过路由器广播 (RA) 获取的。

`IpPrefixOriginUnchanged`

值: 1

IP 地址前缀应保持不变。当 IP 前缀源的值应保持不变时，在设置单播 IP 接口的属性时，将使用此值。

注意 此枚举值仅在 Windows Vista 及更高版本上可用。

注解

IP_PREFIX_ORIGIN 枚举用于 [IP_ADAPTER_UNICAST_ADDRESS](#) 结构的 PrefixOrigin 成员。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织方式已更改，IP_PREFIX_ORIGIN 枚举在 *Iptypes.h* 头文件自动包含的 *Nldef.h* 头文件中定义。若要使用 IP_PREFIX_ORIGIN 枚举，必须在 *Iptypes.h* 头文件之前包含 *Winsock2.h* 头文件。

要求

 展开表

最低受支持的客户端 Windows XP [仅限桌面应用]

最低受支持的服务器 Windows Server 2003 [仅限桌面应用]

标头 *nldef.h* (包括 Windows 8 上的 *Iphlpapi.h*、Windows Server 2008 R2、Windows 7、Windows Server 2008 Windows Vista)

请参阅

[IP_ADAPTER_UNICAST_ADDRESS](#)

反馈

此页面是否有帮助?

是

否

NL_SUFFIX_ORIGIN 枚举 (nldef.h)

项目2023/08/25

[IP_SUFFIX_ORIGIN 枚举指定 IPv4 或 IPv6 地址后缀的原点，并与 IP_ADAPTER_UNICAST_ADDRESS 结构一起使用。](#)

语法

C++

```
typedef enum {
    NlsoOther = 0,
    NlsoManual,
    NlsoWellKnown,
    NlsoDhcp,
    NlsoLinkLayerAddress,
    NlsoRandom,
    IpSuffixOriginOther = 0,
    IpSuffixOriginManual,
    IpSuffixOriginWellKnown,
    IpSuffixOriginDhcp,
    IpSuffixOriginLinkLayerAddress,
    IpSuffixOriginRandom,
    IpSuffixOriginUnchanged = 1 << 4
} NL_SUFFIX_ORIGIN;
```

常量

[展开表](#)

NlsoOther

值: 0

NlsoManual

NlsoWellKnown

NlsoDhcp

NlsoLinkLayerAddress

NlsoRandom

IpSuffixOriginOther
值: 0
IP 地址后缀由此枚举中定义的源以外的其他源提供。
IpSuffixOriginManual
IP 地址后缀是手动指定的。
IpSuffixOriginWellKnown
IP 地址后缀来自已知源。
IpSuffixOriginDhcp
IP 地址后缀由 DHCP 设置提供。
IpSuffixOriginLinkLayerAddress
IP 地址后缀是从链接层地址获取的。
IpSuffixOriginRandom
IP 地址后缀是从随机源获取的。
IpSuffixOriginUnchanged
值: 1
IP 地址后缀应保持不变。当 IP 后缀原点的值应保持不变时，设置单播 IP 接口的属性时，将使用此值。

注意 此枚举值仅适用于 Windows Vista 及更高版本。

注解

IP_SUFFIX_ORIGIN 枚举在 [IP_ADAPTER_UNICAST_ADDRESS](#) 结构的 SuffixOrigin 成员中使用。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，**IP_SUFFIX_ORIGIN** 枚举在 *Iptypes.h* 头文件自动包含的 *Nldef.h* 头文件中定义。若要使用 **IP_SUFFIX_ORIGIN** 枚举，必须在 *Iptypes.h* 头文件之前包含 *Winsock2.h* 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	nldef.h (包括 Windows 8、Windows Server 2008 R2、Windows 7、Windows Server 2008 Windows Vista)

请参阅

[IP_ADAPTER_UNICAST_ADDRESS](#)

反馈

此页面是否有帮助?

 是

 否

NET_ADDRESS_FORMAT 枚举 (iphlpapi.h)

项目2023/08/24

NET_ADDRESS_FORMAT 枚举指定 [ParseNetworkString](#) 函数返回的网络地址的格式。

语法

C++

```
typedef enum NET_ADDRESS_FORMAT_ {
    NET_ADDRESS_FORMAT_UNSPECIFIED = 0,
    NET_ADDRESS_DNS_NAME,
    NET_ADDRESS_IPV4,
    NET_ADDRESS_IPV6
} NET_ADDRESS_FORMAT;
```

常量

NET_ADDRESS_FORMAT_UNSPECIFIED

值: 0

未指定网络地址的格式。

NET_ADDRESS_DNS_NAME

网络地址的格式为 DNS 名称。

NET_ADDRESS_IPV4

网络地址的格式是 IPv4 的 Internet 标准点十进制表示法中的字符串。

NET_ADDRESS_IPV6

网络地址的格式是 IPv6 的 Internet 标准十六进制编码字符串。

注解

NET_ADDRESS_FORMAT 枚举在 Windows Vista 及更高版本上定义。

[ParseNetworkString](#) 函数返回的 [NET_ADDRESS_INFO](#) 结构。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	iphlpapi.h

另请参阅

[NET_ADDRESS_INFO](#)

[ParseNetworkString](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

NL_NETWORK_CONNECTIVITY_LEVEL_HINT 枚举 (nldef.h)

项目2023/08/25

定义指定有关网络连接级别的提示的常量。

语法

C++

```
typedef enum _NL_NETWORK_CONNECTIVITY_LEVEL_HINT {
    NetworkConnectivityLevelHintUnknown = 0,
    NetworkConnectivityLevelHintNone,
    NetworkConnectivityLevelHintLocalAccess,
    NetworkConnectivityLevelHintInternetAccess,
    NetworkConnectivityLevelHintConstrainedInternetAccess,
    NetworkConnectivityLevelHintHidden
} NL_NETWORK_CONNECTIVITY_LEVEL_HINT;
```

常量

NetworkConnectivityLevelHintUnknown

值: 0

指定未知连接级别的提示。在 Windows (或应用程序容器) 启动时，可能会返回此值，会有一个较短的时间窗口。

NetworkConnectivityLevelHintNone

指定无连接的提示。

NetworkConnectivityLevelHintLocalAccess

指定仅限本地网络访问的提示。

NetworkConnectivityLevelHintInternetAccess

指定本地和 Internet 访问的提示。

NetworkConnectivityLevelHintConstrainedInternetAccess

指定受限 Internet 访问的提示。

此值指示强制门户连接，其中提供对 Web 门户的本地访问，但访问 Internet 需要通过门户提供特定凭据。使用公共位置托管的连接时，通常会遇到这种级别的连接，(例如咖啡店和书店)。

这不保证检测强制网络门户。应注意，当 Windows 将连接级别提示报告为

NetworkConnectivityLevelHintLocalAccess 时，可能会重定向应用程序的网络请求，从而收到与预期不同的响应。其他协议也可能受到影响;例如，HTTPS 可能被重定向，身份验证失败。

NetworkConnectivityLevelHintHidden

指定在正常连接 (隐藏的网络接口的提示，默认情况下，应用程序) 无法访问该接口。这可能是因为该网络上不允许任何数据包 (例如，适配器标记自身 NCF_HIDDEN)，或者默认情况下(，)路由在该接口上被忽略 (例如，当连接 WiFi) 时，手机网络处于隐藏状态。

要求

最低受支持的客户端 Windows 10，版本 2004 (10.0;内部版本 19041)

最低受支持的服务器 Windows Server 版本 2004 (10.0;内部版本 19041)

标头 nldef.h (包括 iphlpapi.h)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

NL_NETWORK_CONNECTIVITY_COST_HINT 枚举 (nldef.h)

项目2023/08/25

定义指定有关网络连接使用费的提示的常量。

语法

C++

```
typedef enum _NL_NETWORK_CONNECTIVITY_COST_HINT {
    NetworkConnectivityCostHintUnknown = 0,
    NetworkConnectivityCostHintUnrestricted,
    NetworkConnectivityCostHintFixed,
    NetworkConnectivityCostHintVariable
} NL_NETWORK_CONNECTIVITY_COST_HINT;
```

常量

`NetworkConnectivityCostHintUnknown`

值: 0

指定成本信息不可用的提示。

`NetworkConnectivityCostHintUnrestricted`

指定连接无限制且具有无限制使用费和容量约束的提示。

`NetworkConnectivityCostHintFixed`

指定使用连接不受特定限制的提示。

`NetworkConnectivityCostHintVariable`

指定按字节对连接收费的提示。

要求

最低受支持的客户端

Windows 10, 版本 2004 (10.0; 内部版本 19041)

最低受支持的服务器

Windows Server 版本 2004 (10.0; 内部版本 19041)

标头

nldef.h (包括 iphlpapi.h)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

SCOPE_LEVEL枚举 (ws2def.h)

项目2023/08/28

SCOPE_LEVEL 枚举与 [IP_ADAPTER_ADDRESSES](#) 结构一起使用，以标识 IPv6 地址的范围级别。

语法

C++

```
typedef enum {
    ScopeLevelInterface = 1,
    ScopeLevelLink = 2,
    ScopeLevelSubnet = 3,
    ScopeLevelAdmin = 4,
    ScopeLevelSite = 5,
    ScopeLevelOrganization = 8,
    ScopeLevelGlobal = 14,
    ScopeLevelCount = 16
} SCOPE_LEVEL;
```

常量

 展开表

`ScopeLevelInterface`

值: 1

范围为接口级别。

`ScopeLevelLink`

值: 2

范围为链接级别。

`ScopeLevelSubnet`

值: 3

范围为子网级别。

`ScopeLevelAdmin`

值: 4

范围为管理级别。

`ScopeLevelSite`

值: 5

范围为站点级别。

`ScopeLevelOrganization`

值: 8

范围为组织级别。

`ScopeLevelGlobal`

值: 14

范围为全局。

`ScopeLevelCount`

值: 16

注解

`SCOPE_LEVEL` 枚举在 [IP_ADAPTER_ADDRESSES](#) 结构的 `ZoneIndices` 成员中使用。

在 Windows Vista 和更高版本以及 Microsoft Windows 软件开发工具包 (SDK) 上, 头文件的组织已更改, `SCOPE_LEVEL` 枚举类型在 `Ws2def.h` 头文件中定义。请注意, `Ws2def.h` 头文件会自动包含在 `Winsock2.h` 中, 永远不应直接使用。

要求

[+] 展开表

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	<code>ws2def.h</code> (包括 <code>Winsock2.h</code>)

另请参阅

[IP_ADAPTER_ADDRESSES](#)

反馈

此页面是否有帮助?

是

否

TCP_BOOLEAN_OPTIONAL 枚举 (tcppestats.h)

项目2023/08/26

TCP_BOOLEAN_OPTIONAL 枚举定义调用方在更新 TCP 连接的读/写信息中的成员时可以指定的状态。

语法

C++

```
typedef enum _TCP_BOOLEAN_OPTIONAL {
    TcpBoolOptDisabled = 0,
    TcpBoolOptEnabled,
    TcpBoolOptUnchanged = -1
} TCP_BOOLEAN_OPTIONAL, *PTCP_BOOLEAN_OPTIONAL;
```

常量

TcpBoolOptDisabled

值: 0

应禁用 选项。

TcpBoolOptEnabled

应启用 选项。

TcpBoolOptUnchanged

值: -1

选项应保持不变。

注解

TCP_BOOLEAN_OPTIONAL 枚举在 Windows Vista 及更高版本上定义。

使用对 [SetPerTcp6ConnectionEStats](#) 和 [SetPerTcpConnectionEStats](#) 函数的调用启用和禁用 TCP 连接上的扩展统计信息集合，其中指定的扩展统计信息类型是 [TCP_ESTATS_TYPE](#) 枚举类型中的值之一。TCP_BOOLEAN_OPTIONAL 枚举中的值用于指定应如何更新 [TCP_ESTATS_BANDWIDTH_RW_v0](#) 结构中的成员，以启用或禁用 TCP 连接上的扩展统计信息，以便估计带宽。

要求

最低受支持的客户端	
	Windows Vista [仅限桌面应用]
最低受支持的服务器	
	Windows Server 2008 [仅限桌面应用]
标头	
	tcpstats.h

另请参阅

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_BANDWIDTH_RW_v0](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_TYPE 枚举 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_TYPE枚举定义请求或正在设置的 TCP 连接的扩展统计信息的类型。

语法

C++

```
typedef enum {
    TcpConnectionEstatsSynOpts,
    TcpConnectionEstatsData,
    TcpConnectionEstatsSndCong,
    TcpConnectionEstatsPath,
    TcpConnectionEstatsSendBuff,
    TcpConnectionEstatsRec,
    TcpConnectionEstatsObsRec,
    TcpConnectionEstatsBandwidth,
    TcpConnectionEstatsFineRtt,
    TcpConnectionEstatsMaximum
} TCP_ESTATS_TYPE, *PTCP_ESTATS_TYPE;
```

常量

[+] 展开表

TcpConnectionEstatsSynOpts

此值指定 TCP 连接的 SYN 交换信息。

此枚举值仅提供只读静态信息。

TcpConnectionEstatsData

此值指定 TCP 连接的扩展数据传输信息。

此枚举值只能使用只读动态信息和读/写信息。

TcpConnectionEstatsSndCong

此值指定 TCP 连接的发送方拥塞。

) (只读静态、只读动态和读/写信息的所有三种类型的信息都可用于此枚举值。

TcpConnectionEstatsPath

此值指定 TCP 连接的扩展路径度量信息。 此信息用于推断段

对从本地发送方到远程的路径重新排序接收机。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsSendBuff`

此值指定 TCP 连接的扩展输出队列信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsRec`

此值指定 TCP 连接的扩展本地接收器信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsObsRec`

此值指定 TCP 连接的扩展远程接收器信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsBandwidth`

此值指定带宽上的 TCP 连接的带宽估计统计信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsFineRtt`

此值指定 TCP 连接的精细往返时间 (RTT) 估计统计信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsMaximum`

`TCP_ESTATS_TYPE_STATE` 枚举类型的最大值。对于 TCP 连接的可能类型的扩展统计信息，这不是法律值。

注解

`TCP_ESTATS_TYPE` 枚举在 Windows Vista 及更高版本上定义。

`GetPerTcp6ConnectionEStats` 和 `GetPerTcp6ConnectionEStats` 函数旨在使用 TCP 诊断网络和应用程序中的性能问题。如果基于网络的应用程序性能不佳，TCP 可以确定瓶颈是在发送方、接收方还是网络本身。如果瓶颈在网络中，TCP 可以提供有关其性质的特定信息。

`GetPerTcp6ConnectionEStats` 和 `GetPerTcp6ConnectionEStats` 函数用于根据使用 `TCP_ESTATS_TYPE` 枚举类型中的值之一指定的扩展统计信息的类型检索 TCP 连接的扩展统计信息。使用对 `SetPerTcp6ConnectionEStats` 和 `SetPerTcpConnectionEStats` 函数的调

用启用和禁用 TCP 连接上的扩展统计信息集合，其中指定的扩展统计信息的类型是 **TCP_ESTATS_TYPE** 枚举类型中的值之一。

要求

 展开表

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcppestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_BANDWIDTH_ROD_v0](#)

[TCP_ESTATS_BANDWIDTH_RW_v0](#)

[TCP_ESTATS_DATA_ROD_v0](#)

[TCP_ESTATS_DATA_RW_v0](#)

[TCP_ESTATS_FINE_RTT_ROD_v0](#)

[TCP_ESTATS_FINE_RTT_RW_v0](#)

[TCP_ESTATS_OBS_REC_ROD_v0](#)

[TCP_ESTATS_OBS_REC_RW_v0](#)

[TCP_ESTATS_PATH_ROD_v0](#)

[TCP_ESTATS_PATH_RW_v0](#)

[TCP_ESTATS_REC_ROD_v0](#)

TCP_ESTATS_REC_RW_v0

TCP_ESTATS_SEND_BUFF_ROD_v0

TCP_ESTATS_SEND_BUFF_RW_v0

TCP_ESTATS SND CONG ROD_v0

TCP_ESTATS SND CONG ROS_v0

TCP_ESTATS SND CONG RW_v0

TCP_ESTATS_SYN_OPTS_ROS_v0

反馈

此页面是否有帮助?

 是

 否

TCP_SOFT_ERROR 枚举 (tcppestats.h)

项目2023/08/26

TCP_SOFT_ERROR枚举定义 TCP 连接上记录的非致命或软错误的原因。

语法

C++

```
typedef enum {
    TcpErrorNone = 0,
    TcpErrorBelowDataWindow,
    TcpErrorAboveDataWindow,
    TcpErrorBelowAckWindow,
    TcpErrorAboveAckWindow,
    TcpErrorBelowTsWindow,
    TcpErrorAboveTsWindow,
    TcpErrorDataChecksumError,
    TcpErrorDataLengthError,
    TcpErrorMaxSoftError
} TCP_SOFT_ERROR, *PTCP_SOFT_ERROR;
```

常量

[+] 展开表

TcpErrorNone

值: 0

未发生软错误。

TcpErrorBelowDataWindow

段中的所有数据都如下所示

发送未确认 (SND。 UNA) 序列号。对于保持连接和零窗口探测，此软错误是正常的。

TcpErrorAboveDataWindow

段中的某些数据位于上面

发送窗口 (SND。 WND) 大小。此软错误表示存在实现 bug 或可能攻击。

TcpErrorBelowAckWindow

在 SND 下方收到 ACK。UNA 序列号。此软错误指示返回路径正在对 ACK 重新排序。

TcpErrorAboveAckWindow

对于我们尚未发送的数据，已收到 ACK。
此软错误表示存在实现 bug 或可能的攻击。

TcpErrorBelowTsWindow

段上的时间戳回显回复 (TSecr) 早于
当前 TS。最近 (时间戳，每当
段) 发送。此错误适用于使用由 RFC 1323 中的 IETF 定义的 TCP 时间戳选项 (TSopt) 的 TCP 连接。
有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc1323.txt>。此软错误对于保护防止包装的极少数情况是正常的
PAWS (序列号
机制检测网络重新排序的数据。

TcpErrorAboveTsWindow

段上的 TSecr 比
当前 TS。最近。此软错误表示实现 bug 或
可能的攻击。

TcpErrorDataChecksumError

收到错误的 TCP 校验和。请注意，此值
在本质上是脆弱的，因为标头字段用于
标识连接可能已损坏。

TcpErrorDataLengthError

发生数据长度错误。

此值未在 TCP 扩展统计信息 MIB 上的 IETF 草稿 RFC 中定义。

TcpErrorMaxSoftError

TCP_SOFT_ERROR_STATE 枚举类型的可能最大值。由于 TCP 连接出现软错误的原因，这不是法定
值。

注解

TCP_SOFT_ERROR 枚举在 Windows Vista 及更高版本上定义。

此枚举中的值在 TCP 扩展统计信息 MIB 上的 IETF 草稿 RFC 中定义。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4898.txt>。

要求

 展开表

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpstats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

TCP_TABLE_CLASS 枚举 (iprtrmib.h)

项目2023/08/24

TCP_TABLE_CLASS枚举定义用于指示调用 [GetExtendedTcpTable](#) 返回的表类型的值集。

语法

C++

```
typedef enum _TCP_TABLE_CLASS {
    TCP_TABLE_BASIC_LISTENER,
    TCP_TABLE_BASIC_CONNECTIONS,
    TCP_TABLE_BASIC_ALL,
    TCP_TABLE_OWNER_PID_LISTENER,
    TCP_TABLE_OWNER_PID_CONNECTIONS,
    TCP_TABLE_OWNER_PID_ALL,
    TCP_TABLE_OWNER_MODULE_LISTENER,
    TCP_TABLE_OWNER_MODULE_CONNECTIONS,
    TCP_TABLE_OWNER_MODULE_ALL
} TCP_TABLE_CLASS, *PTCP_TABLE_CLASS;
```

常量

`TCP_TABLE_BASIC_LISTENER`

将返回一个[MIB_TCPTABLE](#) 表，其中包含仅接收本地计算机上的 TCP 终结点) 所有侦听 (。

`TCP_TABLE_BASIC_CONNECTIONS`

包含本地计算机上所有连接的 TCP 终结点的 [MIB_TCPTABLE](#) 表将返回到调用方。

`TCP_TABLE_BASIC_ALL`

包含本地计算机上所有 TCP 终结点的 [MIB_TCPTABLE](#) 表将返回到调用方。

`TCP_TABLE_OWNER_PID_LISTENER`

包含仅接收本地计算机上的 TCP 终结点) 所有侦听 (的[MIB_TCPTABLE_OWNER_PID](#)或[MIB_TCP6TABLE_OWNER_PID](#)将返回到调用方。

`TCP_TABLE_OWNER_PID_CONNECTIONS`

[MIB_TCPTABLE_OWNER_PID](#)或[MIB_TCP6TABLE_OWNER_PID](#)将包含本地计算机上所有连接的 TCP 终结点的结构返回到调用方。

`TCP_TABLE_OWNER_PID_ALL`

包含本地计算机上的所有 TCP 终结点的[MIB_TCPTABLE_OWNER_PID](#)或[MIB_TCP6TABLE_OWNER_PID](#)结构将返回到调用方。

TCP_TABLE_OWNER_MODULE_LISTENER

将 [返回给](#) 调用方 MIB_TCPTABLE_OWNER_MODULE 或 MIB_TCP6TABLE_OWNER_MODULE 结构，该结构包含仅接收本地计算机上的 TCP 终结点) 所有侦听(。

TCP_TABLE_OWNER_MODULE_CONNECTIONS

包含本地计算机上所有连接的 TCP 终结点 [的 MIB_TCPTABLE_OWNER_MODULE 或 MIB_TCP6TABLE_OWNER_MODULE](#) 结构将返回到调用方。

TCP_TABLE_OWNER_MODULE_ALL

包含本地计算机上的所有 TCP 终结点[的 MIB_TCPTABLE_OWNER_MODULE 或 MIB_TCP6TABLE_OWNER_MODULE](#)结构将返回到调用方。

注解

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 中，头文件的组织已更改，TCP_TABLE_CLASS 枚举在 *Iprtrmib.h* 头文件中定义，而不是在 *Iphlpapi.h* 头文件中定义。请注意，*Iprtrmib.h* 头文件会自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Iprtrmib.h* 头文件。

要求

最低受支持的客户端 Windows Vista、Windows XP 和 SP2 [仅限桌面应用]

最低受支持的服务器 Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]

标头 *iprtmb.h* (包括 *Iphlpapi.h*)

另请参阅

[GetExtendedTcpTable](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

TCPIP_OWNER_MODULE_INFO_CLASS 枚举 (iprtrmib.h)

项目2023/08/24

TCPIP_OWNER_MODULE_INFO_CLASS 枚举定义传递给
GetOwnerModuleFromXXXEntry 系列调用的模块信息结构的类型。

语法

C++

```
typedef enum _TCPIP_OWNER_MODULE_INFO_CLASS {
    TCPIP_OWNER_MODULE_INFO_BASIC
} TCPIP_OWNER_MODULE_INFO_CLASS, *PTCPIP_OWNER_MODULE_INFO_CLASS;
```

常量

TCPIP_OWNER_MODULE_INFO_BASIC

TCPIP_OWNER_MODULE_BASIC_INFO 结构传递给 GetOwnerModuleFromXXXEntry 函数。

要求

最低受支持的客户端 Windows Vista、Windows XP SP2 [仅限桌面应用]

最低受支持的服务器 Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]

标头 iprtrmib.h

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获取帮助

UDP_TABLE_CLASS 枚举 (iprtrmib.h)

项目2023/08/24

UDP_TABLE_CLASS 枚举定义用于指示调用 [GetExtendedUdpTable](#) 返回的表类型的值集。

语法

C++

```
typedef enum _UDP_TABLE_CLASS {
    UDP_TABLE_BASIC,
    UDP_TABLE_OWNER_PID,
    UDP_TABLE_OWNER_MODULE
} UDP_TABLE_CLASS, *PUDP_TABLE_CLASS;
```

常量

UDP_TABLE_BASIC

包含本地计算机上所有 UDP 终结点的 [MIB_UDPTABLE](#) 结构将返回到调用方。

UDP_TABLE_OWNER_PID

包含本地计算机上所有 UDP 终结点的 [MIB_UDPTABLE_OWNER_PID](#) 或 [MIB_UDP6TABLE_OWNER_PID](#) 结构将返回到调用方。

UDP_TABLE_OWNER_MODULE

包含本地计算机上所有 UDP 终结点的 [MIB_UDPTABLE_OWNER_MODULE](#) 或 [MIB_UDP6TABLE_OWNER_MODULE](#) 结构将返回到调用方。

注解

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，**UDP_TABLE_CLASS** 枚举在 *Iprtrmib.h* 头文件中定义，而不是在 *Iphlpapi.h* 头文件中定义。请注意，*Iprtrmib.h* 头文件会自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Iprtrmib.h* 头文件。

要求

最低受支持的客户端	Windows Vista、Windows XP 和 SP2 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]
标头	iprtrmib.h (包括 Iphlpapi.h)

另请参阅

[GetExtendedUdpTable](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP 帮助程序函数

项目 · 2023/06/13

以下函数检索和修改本地计算机上的 TCP/IP 传输的配置设置。以下分类列表可帮助确定哪个函数集合最适合给定任务。

- [GetNetworkConnectivityHint](#)
- [GetNetworkConnectivityHintForInterface](#)
- [NotifyNetworkConnectivityHintChange](#)

适配器管理

- [GetAdapterIndex](#)
- [GetAdaptersAddresses](#)
- [GetAdaptersInfo](#)
- [GetPerAdapterInfo](#)
- [GetUniDirectionalAdapterInfo](#)

地址解析协议 (ARP) 管理

- [CreateIplpNetEntry](#)
- [CreateProxyArpEntry](#)
- [DeleteIplpNetEntry](#)
- [DeleteProxyArpEntry](#)
- [FlushIplpNetTable](#)
- [GetIplpNetTable](#)
- [SendARP](#)
- [SetIplpNetEntry](#)

接口转换

- [ConvertInterfaceAliasToLuid](#)
- [ConvertInterfaceGuidToLuid](#)
- [ConvertInterfaceIndexToLuid](#)
- [ConvertInterfaceLuidToAlias](#)
- [ConvertInterfaceLuidToGuid](#)
- [ConvertInterfaceLuidToIndex](#)
- [ConvertInterfaceLuidToNameA](#)
- [ConvertInterfaceLuidToNameW](#)

- [ConvertInterfaceNameToLuidA](#)
- [ConvertInterfaceNameToLuidW](#)
- [if_indextoname](#)
- [if_nametoindex](#)

接口管理

- [FreeInterfaceDnsSettings](#)
- [GetFriendlyIfIndex](#)
- [GetIfEntry](#)
- [GetIfEntry2](#)
- [GetIfEntry2Ex](#)
- [GetIfStackTable](#)
- [GetIfTable](#)
- [GetIfTable2](#)
- [GetIfTable2Ex](#)
- [GetInterfaceDnsSettings](#)
- [GetInterfaceInfo](#)
- [GetInvertedIfStackTable](#)
- [GetIplInterfaceEntry](#)
- [GetIplInterfaceTable](#)
- [GetNumberOfInterfaces](#)
- [InitializeIplInterfaceEntry](#)
- [SetIfEntry](#)
- [SetInterfaceDnsSettings](#)
- [SetIplInterfaceEntry](#)

INTERNET 协议 (IP) 和 Internet 控制消息协议 (ICMP)

- [GetIcmpStatistics](#)
- [GetIpStatistics](#)
- [Icmp6CreateFile](#)
- [Icmp6ParseReplies](#)
- [Icmp6Sendecho2](#)
- [IcmpCloseHandle](#)
- [IcmpCreateFile](#)
- [IcmpParseReplies](#)
- [IcmpSendecho](#)

- [IcmpSendEcho2](#)
- [IcmpSendEcho2Ex](#)
- [SetIpTTL](#)

IP 地址管理

- [AddIpAddress](#)
- [CreateAnycastIpAddressEntry](#)
- [CreateUnicastIpAddressEntry](#)
- [DeleteIpAddress](#)
- [DeleteAnycastIpAddressEntry](#)
- [DeleteUnicastIpAddressEntry](#)
- [GetAnycastIpAddressEntry](#)
- [GetAnycastIpAddressTable](#)
- [GetIpAddrTable](#)
- [GetMulticastIpAddressEntry](#)
- [GetMulticastIpAddressTable](#)
- [GetUnicastIpAddressEntry](#)
- [GetUnicastIpAddressTable](#)
- [InitializeUnicastIpAddressEntry](#)
- [IpReleaseAddress](#)
- [IpRenewAddress](#)
- [NotifyStableUnicastIpAddressTable](#)
- [SetUnicastIpAddressEntry](#)

IP 地址字符串转换

- [RtlIpv4AddressToString](#)
- [RtlIpv4AddressToStringEx](#)
- [RtlIpv4StringToAddress](#)
- [RtlIpv4StringToAddressEx](#)
- [RtlIpv6AddressToString](#)
- [RtlIpv6AddressToStringEx](#)
- [RtlIpv6StringToAddress](#)
- [RtlIpv6StringToAddressEx](#)

IP 邻居地址管理

- [CreateIpNetEntry2](#)

- [DeleteIpNetEntry2](#)
- [FlushIpNetTable2](#)
- [GetIpNetEntry2](#)
- [GetIpNetTable2](#)
- [ResolveIpNetEntry2](#)
- [ResolveNeighbor](#)
- [SetIpNetEntry2](#)

IP 路径管理

- [FlushIpPathTable](#)
- [GetIpPathEntry](#)
- [GetIpPathTable](#)

IP 路由管理

- [CreateIpForwardEntry](#)
- [CreateIpForwardEntry2](#)
- [DeleteIpForwardEntry](#)
- [DeleteIpForwardEntry2](#)
- [EnableRouter](#)
- [GetBestInterface](#)
- [GetBestInterfaceEx](#)
- [GetBestRoute](#)
- [GetBestRoute2](#)
- [GetIpForwardEntry2](#)
- [GetIpForwardTable](#)
- [GetIpForwardTable2](#)
- [GetRTTAndHopCount](#)
- [InitializeIpForwardEntry](#)
- [SetIpForwardEntry](#)
- [SetIpForwardEntry2](#)
- [SetIpStatistics](#)
- [SetIpStatisticsEx](#)
- [UnenableRouter](#)

IP 表内存管理

- [FreeMibTable](#)

IP 实用工具

- ConvertIpv4MaskToLength
- ConvertLengthToIpv4Mask
- CreateSortedAddressPairs
- ParseNetworkString

网络配置

- GetNetworkParams

通知

- CancelMibChangeNotify2
- NotifyAddrChange
- NotifyIpInterfaceChange
- NotifyRouteChange
- NotifyRouteChange2
- NotifyUnicastIpAddressChange

包时间戳

- CaptureInterfaceHardwareCrossTimestamp
- GetInterfaceActiveTimestampCapabilities
- GetInterfaceSupportedTimestampCapabilities
- RegisterInterfaceTimestampConfigChange
- UnregisterInterfaceTimestampConfigChange

永久性端口预留

- CreatePersistentTcpPortReservation
- CreatePersistentUdpPortReservation
- DeletePersistentTcpPortReservation
- DeletePersistentUdpPortReservation
- LookupPersistentTcpPortReservation
- LookupPersistentUdpPortReservation

安全运行状况

- [CancelSecurityHealthChangeNotify](#)
- [NotifySecurityHealthChange](#)

这些函数仅在 Windows Server 2003 上定义。

① 备注

这些功能在 Windows Vista 和 Windows Server 2008 上均不可用。

Teredo IPv6 客户端管理

- [GetTeredoPort](#)
- [NotifyTeredoPortChange](#)
- [NotifyStableUnicastIpAddressTable](#)

传输控制协议 (TCP) 和用户数据报协议 (UDP)

- [GetExtendedTcpTable](#)
- [GetExtendedUdpTable](#)
- [GetOwnerModuleFromTcp6Entry](#)
- [GetOwnerModuleFromTcpEntry](#)
- [GetOwnerModuleFromUdp6Entry](#)
- [GetOwnerModuleFromUdpEntry](#)
- [GetPerTcp6ConnectionEStats](#)
- [GetPerTcpConnectionEStats](#)
- [GetTcpStatistics](#)
- [GetTcpStatisticsEx](#)
- [GetTcpStatisticsEx2](#)
- [GetTcp6Table](#)
- [GetTcp6Table2](#)
- [GetTcpTable](#)
- [GetTcpTable2](#)
- [SetPerTcp6ConnectionEStats](#)
- [SetPerTcpConnectionEStats](#)
- [SetTcpEntry](#)
- [GetUdp6Table](#)
- [GetUdpStatistics](#)
- [GetUdpStatisticsEx](#)
- [GetUdpStatisticsEx2](#)
- [GetUdpTable](#)

弃用的 API

① 备注

这些函数已弃用，Microsoft 不支持这些函数。

- [AllocateAndGetTcpExTableFromStack](#)
- [AllocateAndGetUdpExTableFromStack](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

addIPAddress 函数 (iphlpapi.h)

项目2023/08/24

AddIPAddress 函数将指定的 IPv4 地址添加到指定的适配器。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD AddIPAddress(
    [in]  IPAddr Address,
    [in]  IPMask IpMask,
    [in]  DWORD  IfIndex,
    [out] PULONG NTEContext,
    [out] PULONG NTEInstance
);
```

parameters

[in] Address

要添加到适配器的 IPv4 地址，采用 [IPAddr](#) 结构的形式。

[in] IpMask

Address 参数中指定的 IPv4 地址的子网掩码。 IpMask 参数使用与 [IPAddr](#) 结构相同的格式。

[in] IfIndex

要添加 IPv4 地址的适配器的索引。

[out] NTEContext

指向 ULONG 变量的指针。 成功返回后，此参数指向已添加的 IPv4 地址的 Net Table Entry (NTE) 上下文。 调用方稍后可以在对 [DeleteIPAddress](#) 函数的调用中使用此上下文。

[out] NTEInstance

指向 ULONG 变量的指针。 成功返回后，此参数指向已添加的 IPv4 地址的 NTE 实例。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_DEV_NOT_EXIST	<i>IfIndex</i> 参数指定的适配器不存在。
ERROR_DUP_DOMAINNAME	<i>Address</i> 参数中指定的要添加的 IPv4 地址已存在。
ERROR_GEN_FAILURE	常规故障。对于 <i>Address</i> 参数中指定的某些值（例如通常被视为广播地址的 IPv4 地址），将返回此错误。
ERROR_INVALID_HANDLE	尝试进行函数调用的用户不是管理员。
ERROR_INVALID_PARAMETER	一个或多个参数无效。如果 <i>NTEContext</i> 或 <i>NTEInstance</i> 参数为 NULL，则返回此错误。当 <i>Address</i> 参数中指定的 IP 地址与 <i>IfIndex</i> 参数中指定的接口索引不一致时，也会返回此错误（例如，非环回接口上的环回地址）。
ERROR_NOT_SUPPORTED	运行函数的 Windows 版本不支持函数调用。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`AddIPAddress` 函数用于在本地计算机上添加新的 IPv4 地址条目。`AddIPAddress` 函数添加的 IPv4 地址不是永久性的。只要适配器对象存在，IPv4 地址就存在。重新启动计算机会销毁 IPv4 地址，手动重置网络接口卡 (NIC)。此外，某些 PnP 事件可能会销毁地址。

若要创建保留的 IPv4 地址，可以使用 Windows Management Instrumentation (WMI) 控件中 [Win32_NetworkAdapterConfiguration](#) 类的 `EnableStatic` 方法。`netsh` 命令还可用于创建持久性 IPv4 地址。

有关详细信息，请参阅 Windows 套接字文档中有关 `Netsh.exe` 的文档。

在 Windows Server 2003、Windows XP 和 Windows 2000 上，如果 *Address* 参数中的 IPv4 地址已存在于网络上，则 `AddIPAddress` 函数返回 NO_ERROR 添加的 IPv4 地址为 0.0.0.0。

在 Windows Vista 及更高版本中，如果网络上已存在 *Address* 参数中传递的 IPv4 地址，则 `AddIPAddress` 函数将返回 NO_ERROR 并且添加重复的 IPv4 地址时，

`IP_ADAPTER_UNICAST_ADDRESS` 结构中的 `IP_DAD_STATE` 成员设置为 `IpDadStateDuplicate`。

以后可以通过调用 `DeleteIPAddress` 函数来传递 `AddIPAddress` 函数返回的 `NTEContext` 参数，删除使用 `AddIPAddress` 函数添加的 IPv4 地址。

有关 `IPAddr` 和 `IPMask` 数据类型的信息，请参阅 [Windows 数据类型](#)。若要在点点十进制表示法和 `IPAddr` 格式之间转换 IPv4 地址，请使用 `inet_addr` 和 `inet_ntoa` 函数。

在 Windows Vista 及更高版本上，`CreateUnicastIpAddressEntry` 函数可用于在本地计算机上添加新的单播 IPv4 或 IPv6 地址条目。

示例

以下示例检索 IP 地址表以确定第一个适配器的接口索引，然后将命令行中指定的 IP 地址添加到第一个适配器。然后删除添加的 IP 地址。

C++

```
#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int __cdecl main(int argc, char **argv)
{

    /* Variables used by GetIpAddrTable */
    PMIB_IPADDRTABLE pIPAddrTable;
    DWORD dwSize = 0;
    DWORD dwRetVal = 0;
    IN_ADDR IPAddr;
    DWORD ifIndex;

    /* IPv4 address and subnet mask we will be adding */
    UINT iaIPAddress;
    UINT iaIPMask;

    /* Variables where handles to the added IP are returned */
    ULONG NTEContext = 0;
    ULONG NTEInstance = 0;
```

```

/* Variables used to return error message */
LPVOID lpMsgBuf;

// Validate the parameters
if (argc != 3) {
    printf("usage: %s IPAddress SubnetMask\n", argv[0]);
    exit(1);
}

iaIPAddress = inet_addr(argv[1]);
if (iaIPAddress == INADDR_NONE) {
    printf("usage: %s IPAddress SubnetMask\n", argv[0]);
    exit(1);
}

iaIPMask = inet_addr(argv[2]);
if (iaIPMask == INADDR_NONE) {
    printf("usage: %s IPAddress SubnetMask\n", argv[0]);
    exit(1);
}

// Before calling AddIPAddress we use GetIpAddrTable to get
// an adapter to which we can add the IP.
pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(sizeof(MIB_IPADDRTABLE));
if (pIPAddrTable == NULL) {
    printf("Error allocating memory needed to call GetIpAddrTable\n");
    exit (1);
}
else {
    dwSize = 0;
    // Make an initial call to GetIpAddrTable to get the
    // necessary size into the dwSize variable
    if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==
        ERROR_INSUFFICIENT_BUFFER) {
        FREE(pIPAddrTable);
        pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(dwSize);

    }
    if (pIPAddrTable == NULL) {
        printf("Memory allocation failed for GetIpAddrTable\n");
        exit(1);
    }
}
// Make a second call to GetIpAddrTable to get the
// actual data we want
if ((dwRetVal = GetIpAddrTable(pIPAddrTable, &dwSize, 0)) == NO_ERROR) {
    // Save the interface index to use for adding an IP address
    ifIndex = pIPAddrTable->table[0].dwIndex;
    printf("\n\tInterface Index:\t%ld\n", ifIndex);
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[0].dwAddr;
    printf("\tIP Address: \t\t%s (%lu)%\n", inet_ntoa(IPAddr),
           pIPAddrTable->table[0].dwAddr);
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[0].dwMask;
    printf("\tSubnet Mask: \t\t%s (%lu)%\n", inet_ntoa(IPAddr),
           pIPAddrTable->table[0].dwMask);
}

```

```

    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[0].dwBCastAddr;
    printf("\tBroadCast Address:\t%s (%lu%)\n", inet_ntoa(IPAddr),
           pIPAddrTable->table[0].dwBCastAddr);
    printf("\tReassembly size: \t%lu\n\n",
           pIPAddrTable->table[0].dwReasmSize);
} else {
    printf("Call to GetIpAddrTable failed with error %d.\n", dwRetVal);
    if (pIPAddrTable)
        FREE(pIPAddrTable);
    exit(1);
}

if (pIPAddrTable) {
    FREE(pIPAddrTable);
    pIPAddrTable = NULL;
}

if ((dwRetVal = AddIPAddress(iaIPAddress,
                             iaIPMask,
                             ifIndex,
                             &NTEContext, &NTEInstance)) == NO_ERROR) {
    printf("\tIPv4 address %s was successfully added.\n", argv[1]);
} else {
    printf("AddIPAddress failed with error: %d\n", dwRetVal);

    if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL, dwRetVal,
MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
(LPTSTR) & lpMsgBuf, 0, NULL)) {
        printf("\tError: %s", lpMsgBuf);
        LocalFree(lpMsgBuf);
        exit(1);
    }
}

// Delete the IP we just added using the NTEContext
// variable where the handle was returned
if ((dwRetVal = DeleteIPAddress(NTEContext)) == NO_ERROR) {
    printf("\tIPv4 address %s was successfully deleted.\n", argv[1]);
} else {
    printf("\tDeleteIPAddress failed with error: %d\n", dwRetVal);
    exit(1);
}

exit(0);
}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateUnicastIpAddressEntry](#)

[DeleteIpAddress](#)

[GetAdapterIndex](#)

[GetIpAddrTable](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IPAddr](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

AllocateAndGetTcpExTableFromStack 函数 (iphlpapi.h)

项目2023/08/24

[从 Windows Vista 起，此函数不再可供使用。请改用 [GetTcpTable](#) 或 [GetExtendedTcpTable](#) 函数检索 TCP 连接表。]

AllocateAndGetTcpExTableFromStack 函数检索 TCP 连接表，并从本地堆分配内存以存储表。

语法

C++

```
DWORD AllocateAndGetTcpExTableFromStack(
    [out] PVOID *ppTcpTable,
    [in]  BOOL   bOrder,
    [in]  HANDLE hHeap,
    [in]  DWORD  dwFlags,
    [in]  DWORD  dwFamily
);
```

parameters

`[out] ppTcpTable`

指向函数返回后包含 TCP 连接表的不透明数据的地址的指针。

`[in] bOrder`

如果为 true，则对 *ppTcpTable* 中返回的表中的 TCP 连接条目进行排序;如果为 false，则不为。

`[in] hHeap`

要从中分配存储表的内存的堆的句柄。

`[in] dwFlags`

指示特定堆分配控制行为的一个或多个标志。

`[in] dwFamily`

表中 TCP 地址的系列。

值	含义
AF_INET	检索 IPv4 TCP 地址。
AF_INET6	检索 IPv6 TCP 地址。

返回值

如果函数成功，则返回ERROR_SUCCESS。

如果该函数失败，它将从 winerror.h 返回一个函数。

注解

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 中，AllocateAndGetTcpExTableFromStack 的函数原型仍在 Iphlpapi.h 头文件中定义，以便继续支持 Windows Server 2003 和 Windows XP。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[AllocateAndGetUdpExTableFromStack](#)

[GetExtendedTcpTable](#)

[GetTcpTable](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

AllocateAndGetUdpExTableFromStack 函数 (iphlpapi.h)

项目2023/08/24

[从 Windows Vista 起，此函数不再可供使用。请改用 [GetUdpTable](#) 或 [GetExtendedUdpTable](#) 函数检索 UDP 连接表。]

AllocateAndGetUdpExTableFromStack 函数检索 UDP 连接表，并从本地堆分配内存以存储表。

语法

C++

```
DWORD AllocateAndGetUdpExTableFromStack(  
    [out] PVOID *ppUdpTable,  
    [in]  BOOL   bOrder,  
    [in]  HANDLE hHeap,  
    [in]  DWORD  dwFlags,  
    [in]  DWORD  dwFamily  
) ;
```

parameters

[out] ppUdpTable

指向函数返回后包含 UDP 连接表的不透明数据的地址的指针。

[in] bOrder

如果为 true，则对 *ppUDPTable* 中返回的表中的 UDP 连接条目进行排序;如果为 false，则不为。

[in] hHeap

要从中分配存储表的内存的堆的句柄。

[in] dwFlags

指示特定堆分配控制行为的一个或多个标志。

[in] dwFamily

表中 UDP 地址的系列。

值	含义
AF_INET	检索 IPv4 UDP 地址。
AF_INET6	检索 IPv6 UDP 地址。

返回值

如果函数成功，则返回ERROR_SUCCESS。

如果该函数失败，它将从 winerror.h 返回一个函数。

注解

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 中，AllocateAndGetUdpExTableFromStack 的函数原型仍在 Iphlpapi.h 头文件中定义，以便继续支持 Windows Server 2003 和 Windows XP。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[AllocateAndGetTcpExTableFromStack](#)

[GetExtendedUdpTable](#)

[GetUdpTable](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

CancelIPChangeNotify 函数 (iphlpapi.h)

项目2023/08/24

CancelIPChangeNotify 函数取消之前请求的 IPv4 地址和路由更改的通知，并成功调用 NotifyAddrChange 或 NotifyRouteChange 函数。

语法

C++

```
IPHLPAPI_DLL_LINKAGE BOOL CancelIPChangeNotify(
    [in] LPOVERLAPPED notifyOverlapped
);
```

parameters

[in] notifyOverlapped

指向上次调用 NotifyAddrChange 或 NotifyRouteChange 中使用的 OVERLAPPED 结构的指针。

返回值

无

备注

The

CancelIPChangeNotify 函数取消注册以前在本地计算机上请求的 IPv4 地址或路由更改的更改通知。这些注册通知的请求是通过调用 NotifyAddrChange 或 NotifyRouteChange 函数发出的。

上次调用其中一个通知函数中使用的 OVERLAPPED 结构将传递到 *notifyOverlapped* 参数中的 CancelIPChangeNotify 函数，以取消通知的注册。

如果未找到通知请求或传递了无效的 *notifyOverlapped* 参数，则 CancelIPChangeNotify 可能会返回 FALSE。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[IP 帮助程序函数参考](#)

[NotifyAddrChange](#)

[NotifyRouteChange](#)

[OVERLAPPED](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

CancelMibChangeNotify2 函数 (netioapi.h)

项目2023/08/25

CancelMibChangeNotify2 函数取消注册 IP 接口更改、IP 地址更改、IP 路由更改、Teredo 端口更改以及单播 IP 地址表稳定且可检索的更改通知。

语法

C++

```
IPHLPAPI_DLL_LINKAGE NETIOAPI_API CancelMibChangeNotify2(
    [in] HANDLE NotificationHandle
);
```

parameters

[in] NotificationHandle

从通知注册或检索函数返回的句柄，用于指示要取消的通知。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>NotificationHandle</i> 参数是 NULL 指针，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

CancelMibChangeNotify2 函数在 Windows Vista 及更高版本上定义。

The

取消取消注册之前针对本地计算机上的 IP 接口更改、IP 地址更改、IP 路由更改或 Teredo

端口更改请求的更改通知的 `CancelMibChangeNotify2` 函数。这些请求通过调用 `NotifyIplInterfaceChange`、`NotifyUnicastIpAddressChange`、`NotifyRouteChange2` 或 `NotifyTeredoPortChange` 发出。The `CancelMibChangeNotify2` 函数还会取消当单播 IP 地址表在本地计算机上稳定且可检索时收到通知的先前请求。此请求是通过调用 `NotifyStableUnicastIpAddressTable` 函数发出的。

返回到这些通知函数的 `NotificationHandle` 参数将传递给 `CancelMibChangeNotify2`，以取消通知的注册，或取消挂起的请求以检索稳定的单播 IP 地址表。

应用程序无法从当前正在为同一 `NotificationHandle` 参数执行通知回调函数的线程的上下文调用 `CancelMibChangeNotify2` 函数。否则，执行该回调的线程将导致死锁。因此，不能在通知回调例程中直接调用 `CancelMibChangeNotify2` 函数。在更一般的情况下，执行 `CancelMibChangeNotify2` 函数的线程不能拥有执行通知回调操作的线程将等待的资源，因为这将导致类似的死锁。应从其他线程调用 `CancelMibChangeNotify2` 函数，接收通知回调的线程不依赖于该线程。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[NotifyIplInterfaceChange](#)

[NotifyRouteChange2](#)

[NotifyStableUnicastIpAddressTable](#)

[NotifyTeredoPortChange](#)

[NotifyUnicastIpAddressChange](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

CaptureInterfaceHardwareCrossTimestamp 函数 (iphlpapi.h)

项目 2023/08/24

检索网络适配器的跨时间戳信息。

有关详细信息和代码示例，请参阅 [数据包时间戳](#)。

① 重要

在 Windows 10 版本 2004 (10.0;内部版本 19041) 及更早版本保留此函数供系统使用，不应从代码中调用它。在更高版本中，[支持此函数](#)。

语法

C++

```
IPHLAPI_DLL_LINKAGE DWORD CaptureInterfaceHardwareCrossTimestamp(
    const NET_LUID                *InterfaceLuid,
    PINTERFACE_HARDWARE_CROSSTIMESTAMP CrossTimestamp
);
```

parameters

InterfaceLuid

类型: `_In_ CONST NET_LUID*`

网络本地唯一标识符 (要从中检索交叉时间戳的网络适配器的 LUID)。

CrossTimestamp

类型: `_Inout_ PINTERFACE_HARDWARE_CROSSTIMESTAMP`

时间戳由网络适配器以 `INTERFACE_HARDWARE_CROSSTIMESTAMP` 对象的形式返回。

返回值

类型: `DWORD`

指示成功或失败的 DWORD 返回代码。

要求

最低受支持的客户端	Windows 11 (内部版本 10.0.22000.194)
最低受支持的服务器	Windows Server 2022
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[包时间戳](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

ConvertInterfaceAliasToLuid 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceAliasToLuid 函数将网络接口的接口别名转换为接口的本地唯一标识符 (LUID)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceAliasToLuid(
    [in] const WCHAR *InterfaceAlias,
    [out] PNET_LUID   InterfaceLuid
);
```

parameters

[in] InterfaceAlias

指向以 NULL 结尾的 Unicode 字符串的指针，其中包含网络接口的别名。

[out] InterfaceLuid

指向此接口 NET_LUID 的指针。

返回值

成功后，ConvertInterfaceAliasToLuid 返回NO_ERROR。任何非零返回值都表示失败，InterfaceLuid 参数中返回 NULL。

错误代码	含义
ERROR_INVALID_PARAMETER	某个参数无效。如果 InterfaceAlias 或 InterfaceLuid 参数为 NULL，或者 InterfaceAlias 参数无效，则返回此错误。

注解

ConvertInterfaceAliasToLuid 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceAliasToLuid` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

convertInterfaceGuidToLuid 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceGuidToLuid 函数将网络接口的全局唯一标识符 (GUID) 转换为接口的本地唯一标识符 (LUID)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceGuidToLuid(
    [in] const GUID *InterfaceGuid,
    [out] PNET_LUID InterfaceLuid
);
```

parameters

[in] InterfaceGuid

指向网络接口 GUID 的指针。

[out] InterfaceLuid

指向此接口 NET_LUID 的指针。

返回值

成功后，ConvertInterfaceGuidToLuid 返回NO_ERROR。任何非零返回值都指示失败，InterfaceLuid 参数中返回 NULL。

错误代码	含义
ERROR_INVALID_PARAMETER	某个参数无效。如果 InterfaceAlias 或 InterfaceLuid 参数为 NULL，或者 InterfaceGuid 参数无效，则返回此错误。

注解

ConvertInterfaceGuidToLuid 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceGuidToLuid` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

convertInterfaceIndexToLuid 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceIndexToLuid 函数将网络接口的本地索引转换为接口的本地唯一标识符 (LUID)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceIndexToLuid(
    [in]  NET_IFINDEX InterfaceIndex,
    [out] PNET_LUID    InterfaceLuid
);
```

parameters

[in] InterfaceIndex

网络接口的本地索引值。

[out] InterfaceLuid

指向此接口 NET_LUID 的指针。

返回值

成功后，ConvertInterfaceIndexToLuid 返回NO_ERROR。任何非零返回值都指示失败，InterfaceLuid 参数中返回 NULL。

错误代码	含义
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果 InterfaceIndex 参数指定的网络接口不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	某个参数无效。如果 InterfaceLuid 参数为 NULL 或 InterfaceIndex 参数无效，则返回此错误。

注解

`ConvertInterfaceIndexToLuid` 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceIndexToLuid` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

ConvertInterfaceLuidToAlias 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceLuidToAlias 函数将网络接口的本地唯一标识符 (LUID) 转换为接口别名。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceLuidToAlias(
    [in] const NET_LUID *InterfaceLuid,
    [out] PWSTR           InterfaceAlias,
    [in]  SIZE_T          Length
);
```

parameters

[in] InterfaceLuid

指向网络接口 [NET_LUID](#) 的指针。

[out] InterfaceAlias

指向缓冲区的指针，用于在函数成功返回时保存包含网络接口别名的以 NULL 结尾的 Unicode 字符串。

[in] Length

InterfaceAlias 参数指向的缓冲区的长度 (以字符为单位)。此值必须足够大，才能容纳网络接口的别名和终止 NULL 字符。最大所需长度为 [NDIS_IF_MAX_STRING_SIZE](#) + 1。

返回值

成功后，ConvertInterfaceLuidToAlias 返回NO_ERROR。任何非零返回值都表示失败。

错误代码

含义

ERROR_INVALID_PARAMETER	某个参数无效。如果 <i>InterfaceLuid</i> 或 <i>InterfaceAlias</i> 参数为 NULL，或者 <i>InterfaceLuid</i> 参数无效，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	没有足够的存储空间可用于处理此命令。如果 <i>InterfaceAlias</i> 参数指向的缓冲区大小不够大，不能保留别名，则返回此错误。

注解

`ConvertInterfaceLuidToAlias` 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceLuidToAlias` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。

在 *Ntddndis.h* 头文件中声明没有终止 NULL 的网络接口别名的最大长度

NDIS_IF_MAX_STRING_SIZE。**NDIS_IF_MAX_STRING_SIZE** 定义为 *Ifdef.h* 头文件中定义的**IF_MAX_STRING_SIZE** 常量。*Ntddndis.h* 和 *Ifdef.h* 头文件将自动包含在 *Iphlpapi.h* 头文件自动包含的 *Netioapi.h* 头文件中。不应直接使用 *Ntddndis.h*、*Ifdef.h* 和 *Netioapi.h* 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	<i>netioapi.h</i> (包括 <i>Iphlpapi.h</i>)
Library	<i>Iphlpapi.lib</i>
DLL	<i>Iphlpapi.dll</i>

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

ConvertInterfaceLuidToGuid 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceLuidToGuid 函数将网络接口的本地唯一标识符 (LUID) 转换为接口的全局唯一标识符 (GUID)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceLuidToGuid(
    [in] const NET_LUID *InterfaceLuid,
    [out] GUID           *InterfaceGuid
);
```

parameters

[in] InterfaceLuid

指向网络接口 [NET_LUID](#) 的指针。

[out] InterfaceGuid

指向此接口的 GUID 的指针。

返回值

成功后，ConvertInterfaceLuidToGuid 将返回NO_ERROR。任何非零返回值都表示失败，并且 *InterfaceGuid* 参数中返回 NULL。

错误代码	含义
ERROR_INVALID_PARAMETER	某个参数无效。如果 <i>InterfaceLuid</i> 或 <i>InterfaceGuid</i> 参数为 NULL，或者 <i>InterfaceLuid</i> 参数无效，则返回此错误。

注解

ConvertInterfaceLuidToGuid 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceLuidToGuid` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

ConvertInterfaceLuidToIndex 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceLuidToIndex 函数将网络接口的本地唯一标识符 (LUID) 转换为接口的本地索引。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceLuidToIndex(
    [in] const NET_LUID *InterfaceLuid,
    [out] PNET_IFINDEX InterfaceIndex
);
```

parameters

[in] InterfaceLuid

指向网络接口 [NET_LUID](#) 的指针。

[out] InterfaceIndex

接口的本地索引值。

返回值

成功后，ConvertInterfaceLuidToIndex 返回NO_ERROR。任何非零返回值都指示失败，并在 InterfaceIndex 参数中返回NET_IFINDEX_UNSPECIFIED。

错误代码	含义
ERROR_INVALID_PARAMETER	某个参数无效。如果 InterfaceLuid 或 InterfaceIndex 参数为 NULL，或者 InterfaceLuid 参数无效，则返回此错误。

注解

ConvertInterfaceLuidToIndex 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceLuidToIndex` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

convertInterfaceLuidToNameA 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceLuidToNameA 函数将网络接口的本地唯一标识符 (LUID) 转换为 ANSI 接口名称。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceLuidToNameA(
    [in] const NET_LUID *InterfaceLuid,
    [out] PSTR           InterfaceName,
    [in] SIZE_T          Length
);
```

parameters

[in] InterfaceLuid

指向网络接口 [NET_LUID](#) 的指针。

[out] InterfaceName

指向缓冲区的指针，用于保存函数成功返回时包含接口名称的 **以 NULL 结尾的 ANSI 字符串**。

[in] Length

InterfaceName 参数指向的缓冲区的长度（以字节为单位）。此值必须足够大，以便容纳接口名称和终止 null 字符。所需最大长度为 [NDIS_IF_MAX_STRING_SIZE](#) + 1。

返回值

成功后，ConvertInterfaceLuidToNameA 返回 [NETIO_ERROR_SUCCESS](#)。任何非零返回值都表示失败。

错误代码

含义

ERROR_INVALID_PARAMETER	某个参数无效。如果 <i>InterfaceLuid</i> 或 <i>InterfaceName</i> 参数为 NULL，或者 <i>InterfaceLuid</i> 参数无效，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	没有足够的存储空间可用于处理此命令。如果 <i>InterfaceName</i> 参数指向的缓冲区大小不够大，不能按 <i>Length</i> 参数中指定的大小来保存接口名称，则返回此错误。

注解

`ConvertInterfaceLuidToNameA` 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceLuidToNameA` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。 `ConvertInterfaceLuidToNameA` 将网络接口 LUID 转换为 ANSI 接口名称。

[ConvertInterfaceLuidToNameW](#) 将网络接口 LUID 转换为 Unicode 接口名称。

在 *Ntddndis.h* 头文件中声明没有终止 NULL 的接口名称的最大长度 **NDIS_IF_MAX_STRING_SIZE**。**NDIS_IF_MAX_STRING_SIZE** 定义为 *Ifdef.h* 头文件中定义的**IF_MAX_STRING_SIZE**常量。*Ntddndis.h* 和 *Ifdef.h* 头文件自动包含在 *Iphlpapi.h* 头文件自动包含的 *Netioapi.h* 头文件中。永远不应直接使用 *Ntddndis.h*、*Ifdef.h* 和 *Netioapi.h* 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

ConvertInterfaceLuidToNameW 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceLuidToNameW 函数将网络接口的 LUID (本地唯一标识符) 转换为 Unicode 接口名称。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceLuidToNameW(
    [in] const NET_LUID *InterfaceLuid,
    [out] PWSTR           InterfaceName,
    [in]  SIZE_T          Length
);
```

parameters

[in] InterfaceLuid

指向网络接口 [NET_LUID](#) 的指针。

[out] InterfaceName

指向缓冲区的指针，用于在函数成功返回时保存包含接口名称的 **以 NULL 结尾的 Unicode 字符串**。

[in] Length

InterfaceName 参数指向的数组中的字符数。此值必须足够大，才能容纳接口名称和终止 null 字符。最大所需长度为 [NDIS_IF_MAX_STRING_SIZE](#) + 1。

返回值

成功后，ConvertInterfaceLuidToNameW 返回 [NETIO_ERROR_SUCCESS](#)。任何非零返回值都表示失败。

错误代码

含义

ERROR_INVALID_PARAMETER	某个参数无效。如果 <i>InterfaceLuid</i> 或 <i>InterfaceName</i> 参数为 NULL，或者 <i>InterfaceLuid</i> 参数无效，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	没有足够的存储空间可用于处理此命令。如果 <i>InterfaceName</i> 参数指向的缓冲区大小不够大，不能保留接口名称，则返回此错误。

注解

`ConvertInterfaceLuidToNameW` 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceLuidToNameW` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。`ConvertInterfaceLuidToNameW` 将网络接口 LUID 转换为 Unicode 接口名称。

`ConvertInterfaceLuidToNameA` 将 ANSI 接口名称转换为 LUID。

在 *Ntddndis.h* 头文件中声明没有终止 NULL 的接口名称的最大长度 `NDIS_IF_MAX_STRING_SIZE`。`NDIS_IF_MAX_STRING_SIZE` 定义为 *Ifdef.h* 头文件中定义的 `IF_MAX_STRING_SIZE` 常量。*Ntddndis.h* 和 *Ifdef.h* 头文件将自动包含在 *Iphlpapi.h* 头文件自动包含的 *Netioapi.h* 头文件中。不应直接使用 *Ntddndis.h*、*Ifdef.h* 和 *Netioapi.h* 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	<code>netioapi.h</code> (包括 <code>Iphlpapi.h</code>)
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

ConvertInterfaceNameToLuidA 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceNameToLuidA 函数将 ANSI 网络接口名称转换为接口的 LUID (本地唯一标识符)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceNameToLuidA(
    [in] const CHAR *InterfaceName,
    [out] NET_LUID    *InterfaceLuid
);
```

parameters

[in] InterfaceName

指向包含网络接口名称的以 NULL 结尾的 ANSI 字符串的指针。

[out] InterfaceLuid

指向此接口 NET_LUID 的指针。

返回值

成功后，ConvertInterfaceNameToLuidA 返回 NETIO_ERROR_SUCCESS。任何非零返回值都表示失败。

错误代码	含义
ERROR_BUFFER_OVERFLOW	ANSI 接口名称的长度无效。如果 <i>InterfaceName</i> 参数超过此参数允许的最大字符串长度，则返回此错误。
ERROR_INVALID_NAME	接口名称无效。如果 <i>InterfaceName</i> 参数包含无效的名称，则返回此错误。
ERROR_INVALID_PARAMETER	某个参数无效。如果 <i>InterfaceLuid</i> 参数为 NULL，则返回

此错误。

注解

`ConvertInterfaceNameToLuidA` 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceNameToLuidA` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。 `ConvertInterfaceNameToLuidA` 将 ANSI 接口名称转换为 LUID。

`ConvertInterfaceNameToLuidW` 将 Unicode 接口名称转换为 LUID。

在 *Ntddndis.h* 头文件中声明没有终止 NULL 的接口名称的最大长度 `NDIS_IF_MAX_STRING_SIZE`。`NDIS_IF_MAX_STRING_SIZE` 定义为 *Ifdef.h* 头文件中定义的 `IF_MAX_STRING_SIZE` 常量。*Ntddndis.h* 和 *Ifdef.h* 头文件将自动包含在 *Iphlpapi.h* 头文件自动包含的 *Netioapi.h* 头文件中。不应直接使用 *Ntddndis.h*、*Ifdef.h* 和 *Netioapi.h* 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	<code>netioapi.h</code> (包括 <code>Iphlpapi.h</code>)
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

ConvertInterfaceNameToLuidW 函数 (netioapi.h)

项目2023/08/25

ConvertInterfaceNameToLuidW 函数将 Unicode 网络接口名称转换为接口的 LUID (本地唯一标识符)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
ConvertInterfaceNameToLuidW(
    [in] const WCHAR *InterfaceName,
    [out] NET_LUID     *InterfaceLuid
);
```

parameters

[in] InterfaceName

指向包含网络接口名称的以 NULL 结尾的 Unicode 字符串的指针。

[out] InterfaceLuid

指向此接口 NET_LUID 的指针。

返回值

成功后，ConvertInterfaceNameToLuidW 返回 NETIO_ERROR_SUCCESS。任何非零返回值都表示失败。

错误代码	含义
ERROR_INVALID_NAME	接口名称无效。如果 <i>InterfaceName</i> 参数包含无效名称或 <i>InterfaceName</i> 参数的长度超过了此参数允许的最大字符串长度，则返回此错误。
ERROR_INVALID_PARAMETER	某个参数无效。如果 <i>InterfaceLuid</i> 参数为 NULL，则返回此错误。

注解

`ConvertInterfaceNameToLuidW` 函数在 Windows Vista 及更高版本上可用。

`ConvertInterfaceNameToLuidW` 函数独立于协议，适用于 IPv6 和 IPv4 协议的网络接口。 `ConvertInterfaceNameToLuidW` 将 Unicode 接口名称转换为 LUID。

`ConvertInterfaceNameToLuidA` 将 ANSI 接口名称转换为 LUID。

在 *Ntddndis.h* 头文件中声明没有终止 NULL 的接口名称的最大长度 **NDIS_IF_MAX_STRING_SIZE**。**NDIS_IF_MAX_STRING_SIZE** 定义为 *Ifdef.h* 头文件中定义的**IF_MAX_STRING_SIZE** 常量。*Ntddndis.h* 和 *Ifdef.h* 头文件将自动包含在 *Iphlpapi.h* 头文件自动包含的 *Netioapi.h* 头文件中。不应直接使用 *Ntddndis.h*、*Ifdef.h* 和 *Netioapi.h* 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[NET_LUID](#)

[if_indextoname](#)

[if_nametoindex](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

ConvertIpv4MaskToLength 函数 (netioapi.h)

项目2023/08/25

ConvertIpv4MaskToLength 函数将 IPv4 子网掩码转换为 IPv4 前缀长度。

语法

C++

```
IPHLPAPI_DLL_LINKAGE NETIOAPI_API ConvertIpv4MaskToLength(
    [in] ULONG Mask,
    [out] PUINT8 MaskLength
);
```

parameters

[in] Mask

IPv4 子网掩码。

[out] MaskLength

指向 `UINT8` 值的指针，用于在函数成功返回时保存 IPv4 前缀长度（以位为单位）。

返回值

成功后，`ConvertIpv4MaskToLength` 返回 `NO_ERROR`。任何非零返回值都表示失败。

错误代码	含义
<code>ERROR_INVALID_PARAMETER</code>	某个参数无效。如果 <code>Mask</code> 参数无效，则返回此错误。

注解

`ConvertIpv4MaskToLength` 函数在 Windows Vista 及更高版本上可用。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertLengthToIpv4Mask](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

convertLengthToIpv4Mask 函数 (netioapi.h)

项目2023/08/25

ConvertLengthToIpv4Mask 函数将 IPv4 前缀长度转换为 IPv4 子网掩码。

语法

C++

```
IPHLPAPI_DLL_LINKAGE NETIOAPI_API ConvertLengthToIpv4Mask(
    [in] ULONG MaskLength,
    [out] PULONG Mask
);
```

parameters

[in] MaskLength

IPv4 前缀长度（以位为单位）。

[out] Mask

指向 LONG 值的指针，用于在函数成功返回时保存 IPv4 子网掩码。

返回值

成功后，ConvertLengthToIpv4Mask 返回 NO_ERROR。任何非零返回值都指示失败，*Mask* 参数设置为 *Ws2def.h* 头文件中定义的INADDR_NONE。

错误代码	含义
ERROR_INVALID_PARAMETER	某个参数无效。如果 <i>MaskLength</i> 参数无效，则返回此错误。

注解

ConvertLengthToIpv4Mask 函数在 Windows Vista 及更高版本上可用。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertIpv4MaskToLength](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

CreateAnycastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

CreateAnycastIpAddressEntry 函数在本地计算机上添加新的 anycast IP 地址条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
CreateAnycastIpAddressEntry(
    [in] const MIB_ANYCASTIPADDRESS_ROW *Row
);
```

parameters

[in] Row

指向 anycast IP 地址条目 的MIB_ANYCASTIPADDRESS_ROW 结构条目的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数指向MIB_ANYCASTIPADDRESS_ROW 的 Address 成员未设置为有效的单播 IPv4 或 IPv6 地址，或者 Row 参数指向的MIB_ANYCASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员均未指定，则返回此错误。
ERROR_NOT_FOUND	找不到指定的接口。如果找不到 Row 参数指向的 MIB_ANYCASTIPADDRESS_ROW 的 InterfaceLuid 或

	InterfaceIndex 成员指定的网络接口，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且已在 Row 参数指向的 MIB_ANYCASTIPADDRESS_ROW 的 Address 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，则也会返回此错误。
ERROR_OBJECT_ALREADY_EXISTS	该对象已经存在。如果 Row 参数指向的 MIB_ANYCASTIPADDRESS_ROW 的 Address 成员是 MIB_ANYCASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的接口上现有 anycast IP 地址的副本，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[CreateAnycastIpAddressEntry](#) 函数在 Windows Vista 及更高版本上定义。

[CreateAnycastIpAddressEntry](#) 函数用于在本地计算机上添加新的 anycast IP 地址条目。

Row 参数指向的 [MIB_ANYCASTIPADDRESS_ROW](#) 结构中的 Address 成员必须初始化为有效的单播 IPv4 或 IPv6 地址和系列。此外，必须将指向 Row 参数的 [MIB_ANYCASTIPADDRESS_ROW](#) 结构中的以下成员中的至少一个初始化为接口：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则此成员用于确定要添加单播 IP 地址的接口。如果没有为 InterfaceLuid 成员设置值，(此成员的值) 设置为零，则接下来使用 InterfaceIndex 成员来确定接口。

调用 [CreateAnycastIpAddressEntry](#) 函数时，将忽略行指向的 [MIB_ANYCASTIPADDRESS_ROW](#) 结构的 Scopeld 成员。Scopeld 成员由添加地址的接口自动确定。

如果 Row 参数指向的 [MIB_ANYCASTIPADDRESS_ROW](#) 的 Address 成员中传递的 anycast IP 地址是接口上现有 anycast IP 地址的副本，则 [CreateAnycastIpAddressEntry](#) 函数将失败。

[CreateAnycastIpAddressEntry](#) 函数只能由以 Administrators 组成员身份登录的用户调用。如果 [CreateAnycastIpAddressEntry](#) 由非管理员组成员的用户调用，则函数调用将失败并返回 [ERROR_ACCESS_DENIED](#)。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文

件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员（RunAs 管理员）才能使此功能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[DeleteAnycastIpAddressEntry](#)

[GetAnycastIpAddressEntry](#)

[GetAnycastIpAddressTable](#)

[IP 帮助程序函数参考](#)

[MIB_ANYCASTIPADDRESS_ROW](#)

[MIB_ANYCASTIPADDRESS_TABLE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

CreateIpForwardEntry 函数 (iphlpapi.h)

项目2023/08/24

CreateIpForwardEntry 函数在本地计算机的 IPv4 路由表中创建路由。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD CreateIpForwardEntry(
    [in] PMIB_IPFORWARDROW pRoute
);
```

parameters

[in] pRoute

指向 [MIB_IPFORWARDROW](#) 结构的指针，该结构指定新路由的信息。调用方必须为此结构的所有成员指定值。调用方必须为 [MIB_IPFORWARDROW](#) 的 dwForwardProto 成员指定 [MIB IPPROTO_NETMGMT](#)。

返回值

如果函数成功，则函数返回 [NO_ERROR](#) (零)。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (运行方式管理员)。
ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。如果 <i>pRoute</i> 参数为 NULL、 MIB_IPFORWARDROW 的 dwForwardProto 成员未设置为 MIB IPPROTO_NETMGMT 、 PMIB_IPFORWARDROW 结构的 dwForwardMask 成员不是有效的 IPv4 子网掩码，或者 MIB_IPFORWARDROW 结构的某个其他成员无效，则返回此错误。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。

注解

路由参数指向的 [MIB_IPFORWARDROW](#) 结构的 dwForwardProto 成员必须设置为 [MIB IPPROTO_NETMGMT](#) 否则 [CreateIpForwardEntry](#) 将失败。 路由协议标识符用于标识指定路由协议的路由信息。 例如， [MIB IPPROTO_NETMGMT](#) 用于通过网络管理（例如 动态主机配置协议 (DHCP)、简单网络管理协议 (SNMP)）或通过调用 [CreateIpForwardEntry](#)、[DeleteIpForwardEntry](#) 或 [SetIpForwardEntry](#) 函数来标识 IP 路由设置的路由信息。

在 Windows Vista 和 Windows Server 2008 上，*pRoute* 参数指向的 [MIB_IPFORWARDROW](#) 结构的 dwForwardMetric1 成员中指定的路由指标表示添加到关联接口的 [MIB_IPINTERFACE_ROW](#) 结构的 [指标](#) 成员中指定的路由指标的组合。 因此，[MIB_IPFORWARDROW](#) 结构的 dwForwardMetric1 成员应等于或大于关联 [MIB_IPINTERFACE_ROW](#) 结构的 Metric 成员。 如果应用程序想要将路由指标设置为 0，则应将 [MIB_IPFORWARDROW](#) 结构的 dwForwardMetric1 成员设置为等于关联 [MIB_IPINTERFACE_ROW](#) 结构的 Metric 成员中指定的接口指标的值。 应用程序可以通过调用 [GetIpInterfaceEntry](#) 函数来检索接口指标。

在 Windows Vista 和 Windows Server 2008 上，[CreateIpForwardEntry](#) 仅适用于具有单个子接口的接口（其中接口 LUID 和子接口 LUID 是相同的）。 [MIB_IPFORWARDROW](#) 结构的 dwForwardIfIndex 成员指定接口。

[CreateIpForwardEntry](#) 当前不使用路由参数指向的 [MIB_IPFORWARDROW](#) 结构中的许多成员。 这些成员包括 dwForwardPolicy、dwForwardType、dwForwardAge、dwForwardNextHopAS、dwForwardMetric2、dwForwardMetric3、dwForwardMetric4 和 dwForwardMetric5。

[CreateIpForwardEntry](#) 创建的新路由将自动具有 DWForwardAge 的默认值 INFINITE。

若要修改 IPv4 路由表中的现有路由，请使用 [SetIpForwardEntry](#) 函数。 若要检索 IPv4 路由表，请调用 [GetIpForwardTable](#) 函数。

在 Windows Vista 及更高版本中，[CreateIpForwardEntry](#) 函数只能由以管理员组成员身份登录的用户调用。 如果 [CreateIpForwardEntry](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。

[CreateIpForwardEntry](#) 函数也可能因为用户帐户控制（Windows Vista 及更高版本上的 UAC）而失败。 如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。 如果应用程序缺少此清单文

件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员（RunAs 管理员）才能使此功能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。

若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

示例

以下示例演示如何将默认网关更改为 NewGateway。只需调用 GetIpForwardTable，更改网关，然后调用 SetIpForwardEntry 不会更改路由，而只会添加新路由。如果由于某种原因存在多个默认网关，此代码将删除它们。请注意，新网关必须可行；否则，TCP/IP 将忽略更改。

注意 执行此代码将更改 IP 路由表，并可能导致网络活动失败。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "iphlpapi.lib")

int main()
{
    // Declare and initialize variables

    PMIB_IPFORWARDTABLE pIpForwardTable = NULL;
    PMIB_IPFORWARDROW pRow = NULL;
    DWORD dwSize = 0;
    BOOL bOrder = FALSE;
    DWORD dwStatus = 0;
    DWORD NewGateway = 0xDDCCBBAA; // this is in host order Ip Address
    AA.BB.CC.DD is DDCCBBAA

    unsigned int i;
```

```

// Find out how big our buffer needs to be.
dwStatus = GetIpForwardTable(pIpForwardTable, &dwSize, bOrder);
if (dwStatus == ERROR_INSUFFICIENT_BUFFER) {
    // Allocate the memory for the table
    if (!(pIpForwardTable = (PMIB_IPFORWARDTABLE) malloc(dwSize))) {
        printf("Malloc failed. Out of memory.\n");
        exit(1);
    }
    // Now get the table.
    dwStatus = GetIpForwardTable(pIpForwardTable, &dwSize, bOrder);
}

if (dwStatus != ERROR_SUCCESS) {
    printf("getIpForwardTable failed.\n");
    if (pIpForwardTable)
        free(pIpForwardTable);
    exit(1);
}
// Search for the row in the table we want. The default gateway has a
destination
// of 0.0.0.0. Notice that we continue looking through the table, but
copy only
// one row. This is so that if there happen to be multiple default
gateways, we can
// be sure to delete them all.
for (i = 0; i < pIpForwardTable->dwNumEntries; i++) {
    if (pIpForwardTable->table[i].dwForwardDest == 0) {
        // We have found the default gateway.
        if (!pRow) {
            // Allocate some memory to store the row in; this is easier
than filling
                // in the row structure ourselves, and we can be sure we
change only the
                // gateway address.
            pRow = (PMIB_IPFORWARDROW) malloc(sizeof
(MIB_IPFORWARDROW));
            if (!pRow) {
                printf("Malloc failed. Out of memory.\n");
                exit(1);
            }
            // Copy the row
            memcpy(pRow, &(pIpForwardTable->table[i]),
sizeof (MIB_IPFORWARDROW));
        }
        // Delete the old default gateway entry.
        dwStatus = DeleteIpForwardEntry(&(pIpForwardTable->table[i]));

        if (dwStatus != ERROR_SUCCESS) {
            printf("Could not delete old gateway\n");
            exit(1);
        }
    }
}
// Set the nexthop field to our new gateway - all the other properties

```

```

of the route will
// be the same as they were previously.
pRow->dwForwardNextHop = NewGateway;

// Create a new route entry for the default gateway.
dwStatus = CreateIpForwardEntry(pRow);

if (dwStatus == NO_ERROR)
    printf("Gateway changed successfully\n");
else if (dwStatus == ERROR_INVALID_PARAMETER)
    printf("Invalid parameter.\n");
else
    printf("Error: %d\n", dwStatus);

// Free resources
if (pIpForwardTable)
    free(pIpForwardTable);
if (pRow)
    free(pRow);

exit(0);
}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[DeleteIpForwardEntry](#)

[GetIpForwardTable](#)

[GetIpInterfaceEntry](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPFORWARDROW](#)

[SetIpForwardEntry](#)

反馈

此页面是否有帮助？

[!\[\]\(e6ea76a964ea17ff2b3b0adf30d086fe_img.jpg\) 是](#)

[!\[\]\(ebd38242175907eb769150eb00eee1ff_img.jpg\) 否](#)

[在 Microsoft Q&A 获得帮助](#)

createIpForwardEntry2 函数 (netioapi.h)

项目2023/08/25

CreateIpForwardEntry2 函数在本地计算机上创建新的 IP 路由条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API CreateIpForwardEntry2(
    [in] const MIB_IPFORWARD_ROW2 *Row
);
```

parameters

[in] Row

指向 IP 路由条目 [MIB_IPFORWARD_ROW2](#) 结构条目的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置的管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，未指定 Row 参数所指向 MIB_IPFORWARD_ROW2 的 DestinationPrefix 成员、Row 参数指向 MIB_IPFORWARD_ROW2 的 NextHop 成员，或者 Row 参数指向的 MIB_IPFORWARD_ROW2 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误未指定参数。如果 MIB_IPFORWARD_ROW2 中指定的 PreferredLifetime 成员大于 ValidLifetime 成员，或者 MIB_IPFORWARD_ROW2 中的 SitePrefixLength 大于 DestinationPrefix 中指定的前缀长度，也会返回此错误。

ERROR_NOT_FOUND	找不到指定的接口。如果找不到 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果指定的接口不支持路由，则返回此错误。如果本地计算机上没有 IPv4 堆栈，并且 AF_INET 在 Row 参数指向的 MIB_IPFORWARD_ROW2 的 DestinationPrefix 成员的地址系列中指定，也会返回此错误。如果本地计算机上没有 IPv6 堆栈，并且为 DestinationPrefix 成员中的地址系列指定了 AF_INET6，也会返回此错误。
ERROR_OBJECT_ALREADY_EXISTS	该对象已经存在。如果 Row 参数指向的 MIB_IPFORWARD_ROW2 的 DestinationPrefix 成员是 MIB_IPFORWARD_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员指定的接口上现有 IP 路由条目的副本，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`CreateIpForwardEntry2` 函数在 Windows Vista 及更高版本上定义。

`CreateIpForwardEntry2` 函数用于在本地计算机上添加新的邻居 IP 地址条目。

`InitializeIpForwardEntry` 函数应用于使用默认值初始化 [MIB_IPFORWARD_ROW2](#) 结构条目的成员。然后，应用程序可以更改要修改的 [MIB_IPFORWARD_ROW2](#) 条目中的成员，然后调用 `CreateIpForwardEntry2` 函数。

Row 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构中的 DestinationPrefix 成员必须初始化为有效的 IPv4 或 IPv6 地址前缀。Row 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构中的 NextHop 成员必须初始化为有效的 IPv4 或 IPv6 地址和系列。此外，[MIB_IPFORWARD_ROW2 结构中](#) 指向 Row 参数的下列成员中至少有一个必须初始化到接口：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则此成员用于确定要添加 IP 路由条目的接口。如果未为 InterfaceLuid 成员设置值（此成员的值设置为零），则接下来使用 InterfaceIndex 成员来确定接口。

Row 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构的 Metric 成员中指定的路由指标偏移量仅代表整个路由指标的一部分。完整指标是添加到关联接口的 [MIB_IPINTERFACE_ROW](#) 结构的 [指标](#) 成员中指定的接口指标的路由指标偏移量的组合。应用程序可以通过调用 `GetIpInterfaceEntry` 函数来检索接口指标。

调用 `CreateIpForwardEntry2` 函数时，将忽略行指向的 [MIB_IPFORWARD_ROW2](#) 结构的 `Age` 和 `Origin` 成员。这些成员由网络堆栈设置，不能使用 `CreateIpForwardEntry2` 函数进行设置。

如果 `Row` 参数所指向 [MIB_IPFORWARD_ROW2](#) 的 `DestinationPrefix` 和 `NextHop` 成员是 `InterfaceLuid` 或 `InterfaceIndex` 成员中指定的接口上现有 IP 路由条目的副本，则 `CreateIpForwardEntry2` 函数将失败。

`CreateIpForwardEntry2` 函数只能由以 Administrators 组成员身份登录的用户调用。如果不是 Administrators 组成员的用户调用 `CreateIpForwardEntry2`，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[DeleteIpForwardEntry2](#)

[GetBestRoute2](#)

[GetIpForwardEntry2](#)

[GetIpForwardTable2](#)

[GetIplInterfaceEntry](#)

InitializeIpForwardEntry

MIB_IPFORWARD_ROW2

MIB_IPFORWARD_TABLE2

MIB_IPINTERFACE_ROW

NotifyRouteChange2

SetIpForwardEntry2

反馈

此页面是否有帮助？

 是

 否

在 Microsoft Q&A 获得帮助

createIpNetEntry 函数 (iphlpapi.h)

项目2023/08/24

CreateIpNetEntry 函数在本地计算机上的 ARP 表中创建地址解析协议 (ARP) 条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD CreateIpNetEntry(
    [in] PMIB_IPNETROW pArpEntry
);
```

parameters

[in] pArpEntry

指向 [MIB_IPNETROW](#) 结构的指针，该结构指定新条目的信息。 调用方必须为此结构的所有成员指定值。

返回值

如果函数成功，则函数返回 NO_ERROR (零)。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。如果 pArpEntry 参数为 NULL， MIB_IPNETROW 的 dwPhysAddrLen 成员设置为零或值大于 8， MIB_IPNETROW 结构的 dwAddr 成员无效，或 MIB_IPNETROW 结构的其他成员之一无效，则返回此错误。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

若要修改现有的 ARP 条目，请使用 [SetIpNetEntry](#) 函数。 若要检索 ARP 表，请调用 [GetIpNetTable](#) 函数。 若要删除现有的 ARP 条目，请调用 [DeleteIpNetEntry](#)。

在 Windows Vista 及更高版本上，[CreateIpNetEntry](#) 函数只能由以 Administrators 组成员身份登录的用户调用。 如果 [CreateIpNetEntry](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。

[CreateIpNetEntry](#) 函数也可能因为用户帐户控制 (Windows Vista 上的 UAC) 而失败。 如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。 如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。
若要成功执行此函数，调用方必须以 Administrators 组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateProxyArpEntry](#)

[DeleteIpNetEntry](#)

[DeleteProxyArpEntry](#)

[FlushIpNetTable](#)

[GetIpNetTable](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPNETROW](#)

[SetIpNetEntry](#)

反馈

此页面是否有帮助?

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

createIpNetEntry2 函数 (netioapi.h)

项目2023/08/25

CreateIpNetEntry2 函数在本地计算机上创建新的邻居 IP 地址条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API CreateIpNetEntry2(
    [in] const MIB_IPNET_ROW2 *Row
);
```

parameters

[in] Row

指向邻居 IP 地址条目 [MIB_IPNET_ROW2](#) 结构条目的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数指向 MIB_IPNET_ROW2 的 Address 成员未设置为有效的单播、任意广播或多播 IPv4 或 IPv6 地址，Row 参数指向的 MIB_IPNET_ROW2 的 PhysicalAddress 和 PhysicalAddressLength 成员未设置为有效的物理地址，则返回此错误。或 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员均未指定。如果在 Address 成员中传递了环回地址，也会返回此错误。
ERROR_NOT_FOUND	找不到指定的接口。如果找不到 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口，则返回此错误。

ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且已在 Row 参数指向的 MIB_IPNET_ROW2 的 Address 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，则也会返回此错误。
ERROR_OBJECT_ALREADY_EXISTS	该对象已经存在。如果 Row 参数指向的 MIB_IPNET_ROW2 的 Address 成员是 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员指定的接口上的现有邻居 IP 地址的副本，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

CreateIpNetEntry2 函数在 Windows Vista 及更高版本上定义。

CreateIpNetEntry2 函数用于在本地计算机上添加新的邻居 IP 地址条目。

Row 参数指向的 MIB_IPNET_ROW2 结构中的 Address 成员必须初始化为有效的单播、任意广播或多播 IPv4 或 IPv6 地址和系列。Row 参数指向的 MIB_IPNET_ROW2 结构中的 PhysicalAddress 和 PhysicalAddressLength 成员必须初始化为有效的物理地址。此外，必须将指向 Row 参数的 MIB_IPNET_ROW2 结构中的以下成员中的至少一个初始化为接口：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则此成员用于确定要添加单播 IP 地址的接口。如果没有为 InterfaceLuid 成员设置值，(此成员的值) 设置为零，则接下来使用 InterfaceIndex 成员来确定接口。

如果 Row 参数指向的 MIB_IPNET_ROW2 的 Address 成员中传递的 IP 地址是接口上现有邻居 IP 地址的副本，则 CreateIpNetEntry2 函数将失败。

CreateIpNetEntry2 函数只能由以管理员组成员身份登录的用户调用。如果 CreateIpNetEntry2 由非管理员组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[DeleteIpNetEntry2](#)

[FlushIpNetTable2](#)

[GetIpNetEntry2](#)

[GetIpNetTable2](#)

[MIB_IPNET_ROW2](#)

[MIB_IPNET_TABLE2](#)

[ResolveIpNetEntry2](#)

[SetIpNetEntry2](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

createPersistentTcpPortReservation 函数 (iphlpapi.h)

项目2023/08/24

CreatePersistentTcpPortReservation 函数为本地计算机上 TCP 端口的连续块创建永久性 TCP 端口预留。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG CreatePersistentTcpPortReservation(
    [in] USHORT StartPort,
    [in] USHORT NumberOfPorts,
    [out] PULONG64 Token
);
```

parameters

[in] StartPort

按网络字节顺序排列的起始 TCP 端口号。

[in] NumberOfPorts

要保留的 TCP 端口号数。

[out] Token

指向在函数成功时返回的端口预留令牌的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的

shell 中运行，因为内置的管理员 (RunAs 管理员)。

ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果在 <i>StartPort</i> 或 <i>NumberOfPorts</i> 参数中传递零，则返回此错误。如果 <i>NumberOfPorts</i> 参数的端口块太大，则也会返回此错误，具体取决于可分配的端口块的端口将超过可分配的最大端口的 <i>StartPort</i> 参数。
ERROR_SHARING_VIOLATION	进程无法访问该文件，因为它正在被另一个进程使用。如果已使用由 <i>StartPort</i> 和 <i>NumberOfPorts</i> 参数指定的 TCP 端口块中的 TCP 端口，则返回此错误。如果 <i>StartPort</i> 和 <i>NumberOfPorts</i> 参数指定的 TCP 端口块的永久性预留与已创建的 TCP 端口块的永久性预留匹配或重叠，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

CreatePersistentTcpPortReservation 函数在 Windows Vista 及更高版本上定义。

CreatePersistentTcpPortReservation 函数用于为 TCP 端口块添加永久性预留。

需要保留端口的应用程序和服务分为两类。第一个类别包括需要特定端口作为其操作的一部分的组件。此类组件通常倾向于在应用程序清单中 (在安装时指定所需的端口，例如)。第二个类别包括在运行时需要任何可用端口或端口块的组件。

这两个类别对应于特定和通配符端口预留请求。特定预留请求可以是永久性的或运行时的，而通配符端口预留请求仅在运行时受支持。

CreatePersistentTcpPortReservation 函数使应用程序或服务能够保留 TCP 端口的永久性块。永久性 TCP 端口预留记录在 Windows 中 TCP 模块的持久存储中。

调用方通过指定所需的端口数以及是否需要特定范围来获取永久性端口预留。如果请求可以满足，**CreatePersistentTcpPortReservation** 函数将返回唯一不透明 ULONG64 令牌，该令牌随后标识预留。可以通过调用 [DeletePersistentTcpPortReservation](#) 函数释放永久性 TCP 端口预留。请注意，每次重启系统时，给定的永久性 TCP 端口预留的令牌可能会更改。

Windows 不会为使用这些函数获取的永久性预留实现组件间安全性。这意味着，如果向某个组件授予获取任何永久性端口预留的能力，该组件将自动获得使用授予系统上任何其他组件的任何永久性端口预留的能力。对运行时预留强制实施进程级安全性，但此类控制不能扩展到使用 **CreatePersistentTcpPortReservation** 或 [CreatePersistentUdpPortReservation](#) 函数创建的永久性端口预留。

获取永久性 TCP 端口预留后，应用程序可以通过打开 TCP 套接字、调用 [WSAOctl](#) 函数（指定 [SIO_ASSOCIATE_PORT_RESERVATION](#) IOCTL 并传递预留令牌，然后再对套接字上的 [绑定](#) 函数发出调用）从 TCP 端口预留请求端口分配。

[SIO_ACQUIRE_PORT_RESERVATION](#) IOCTL 可用于请求 TCP 或 UDP 端口块的运行时预留。对于运行时端口预留，端口池要求从向其授予预留的套接字的进程使用预留。运行时端口预留的持续时间仅为调用 [SIO_ACQUIRE_PORT_RESERVATION](#) IOCTL 的套接字的生存期。相比之下，使用 [CreatePersistentTcpPortReservation](#) 函数创建的永久性端口预留可由能够获取永久性预留的任何进程使用。

[CreatePersistentTcpPortReservation](#) 函数只能由以 Administrators 组成员身份登录的用户调用。如果不是 Administrators 组成员的用户调用

[CreatePersistentTcpPortReservation](#)，则函数调用将失败并返回 [ERROR_ACCESS_DENIED](#)。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 [requestedExecutionLevel](#) 设置为 [requireAdministrator](#)。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

示例

以下示例创建一个永久性 TCP 端口预留，然后创建一个套接字并从端口预留中分配一个端口，然后关闭该套接字并删除 TCP 端口预留。

此示例必须由管理员组成员的用户运行。运行此示例的最简单方法是使用增强的 shell，因为内置管理员 (RunAs 管理员)。

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <Windows.h>
#include <winsock2.h>
#include <mstcpip.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>
```

```
// Need to link with iphlpapi.lib
#pragma comment(lib, "iphlpapi.lib")

// Need to link with Ws2_32.lib for Winsock functions
#pragma comment(lib, "ws2_32.lib")

int wmain(int argc, WCHAR ** argv)
{

    // Declare and initialize variables

    int startPort = 0;           // host byte order
    int numPorts = 0;
    USHORT startPortns = 0;      // Network byte order
    ULONG64 resToken = { 0 };

    unsigned long status = 0;

    WSADATA wsaData = { 0 };
    int iResult = 0;

    SOCKET sock = INVALID_SOCKET;
    int iFamily = AF_INET;
    int iType = SOCK_STREAM;
    int iProtocol = IPPROTO_TCP;

    DWORD bytesReturned = 0;

    // Note that the sockaddr_in struct works only with AF_INET not AF_INET6
    // An application needs to use the sockaddr_in6 for AF_INET6
    sockaddr_in service;
    sockaddr_in sockName;
    int nameLen = sizeof(sockName);

    // Validate the parameters
    if (argc != 3) {
        wprintf(L"usage: %s <Starting Port> <Number of Ports>\n",
               argv[0]);
        wprintf(L"Creates a persistent TCP port reservation\n");
        wprintf(L"Example usage:\n");
        wprintf(L"    %s 5000 20\n", argv[0]);
        wprintf(L"    where StartPort=5000 NumPorts=20");
        return 1;
    }

    startPort = _wtoi(argv[1]);
    if (startPort < 0 || startPort > 65535) {
        wprintf(L"Starting point must be either 0 or between 1 and
65,535\n");
        return 1;
    }
    startPortns = htons((USHORT) startPort);

    numPorts = _wtoi(argv[2]);
}
```

```

if (numPorts < 0) {
    wprintf(L"Number of ports must be a positive number\n");
    return 1;
}

status =
    CreatePersistentTcpPortReservation((USHORT) startPortns, (USHORT)
numPorts,
                                         &resToken);
if (status != NO_ERROR) {
    wprintf(L"CreatePersistentTcpPortReservation returned error: %ld\n",
status);
    return 1;
}

wprintf(L"CreatePersistentTcpPortReservation call succeeded\n");
wprintf(L" Token = %I64d\n", resToken);

// Comment out this block if you don't want to create a socket and
associate it with the
// persistent reservation

// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult != 0) {
    wprintf(L"WSAStartup failed with error = %d\n", iResult);
    // return 1;
}

sock = socket(iFamily, iType, iProtocol);
if (sock == INVALID_SOCKET)
    wprintf(L"socket function failed with error = %d\n",
WSAGetLastError());
else {
    wprintf(L"socket function succeeded\n");

    iResult =
        WSAIoctl(sock, SIO_ASSOCIATE_PORT_RESERVATION, (LPVOID) &
resToken,
                  sizeof (ULONG64), NULL, 0, &bytesReturned, NULL, NULL);
    if (iResult != 0) {
        wprintf
            (L"WSAIoctl(SIO_ASSOCIATE_PORT_RESERVATION) failed with
error = %d\n",
             WSAGetLastError());
    } else {
        wprintf(L"WSAIoctl(SIO_ASSOCIATE_PORT_RESERVATION) succeeded,
bytesReturned = %u\n",
               bytesReturned);

        service.sin_family = AF_INET;
        service.sin_addr.s_addr = INADDR_ANY;
        service.sin_port = 0;

        iResult = bind(sock, (SOCKADDR*) &service, sizeof(service));
    }
}
}

```

```

        if (iResult == SOCKET_ERROR)
            wprintf(L"bind failed with error = %d\n",
WSAGetLastError());
        else {
            wprintf(L"bind succeeded\n");
            iResult = getsockname(sock, (SOCKADDR*) &sockName,
&nameLen);
            if (iResult == SOCKET_ERROR)
                wprintf(L"getsockname failed with error = %d\n",
WSAGetLastError() );
            else {
                wprintf(L"getsockname succeeded\n");
                wprintf(L"Port number allocated = %u\n",
ntohs(sockName.sin_port) );
            }
        }
    }

    if (sock != INVALID_SOCKET) {
        iResult = closesocket(sock);
        if (iResult == SOCKET_ERROR) {
            wprintf(L"closesocket failed with error = %d\n",
WSAGetLastError());
            WSACleanup();
        }
    }
}

// comment out this block of code if you don't want to delete the
reservation just created
status = DeletePersistentTcpPortReservation((USHORT) startPortns,
(USHORT) numPorts);
if (status != NO_ERROR) {
    wprintf(L"DeletePersistentTcpPortReservation returned error: %ld\n",
status);
    return 1;
}
wprintf(L"DeletePersistentTcpPortReservation call succeeded\n");

return 0;
}

```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]

目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreatePersistentUdpPortReservation](#)

[DeletePersistentTcpPortReservation](#)

[DeletePersistentUdpPortReservation](#)

[LookupPersistentTcpPortReservation](#)

[LookupPersistentUdpPortReservation](#)

[SIO_ACQUIRE_PORT_RESERVATION](#)

[SIO_ASSOCIATE_PORT_RESERVATION](#)

[SIO_RELEASE_PORT_RESERVATION](#)

[WSAloctl](#)

[bind](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

createPersistentUdpPortReservation 函数 (iphlpapi.h)

项目2023/03/14

CreatePersistentUdpPortReservation 函数为本地计算机上的 UDP 端口的连续块创建持久性 UDP 端口预留。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG CreatePersistentUdpPortReservation(
    [in] USHORT StartPort,
    [in] USHORT NumberOfPorts,
    [out] PULONG64 Token
);
```

参数

[in] StartPort

按网络字节顺序排列的起始 UDP 端口号。

[in] NumberOfPorts

要保留的 UDP 端口号数。

[out] Token

指向在函数成功时返回的端口预留令牌的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置的管理员 (RunAs 管理员)。

ERROR_INVALID_PARAMETER	传递给函数的参数无效。 如果在 <i>StartPort</i> 或 <i>NumberOfPorts</i> 参数中传递零，则返回此错误。如果 <i>NumberOfPorts</i> 参数的端口块太大，则也会返回此错误，具体取决于可分配的端口块的端口将超过可分配的最大端口的 <i>StartPort</i> 参数。
ERROR_SHARING_VIOLATION	进程无法访问该文件，因为它正在被另一个进程使用。如果已使用由 <i>StartPort</i> 和 <i>NumberOfPorts</i> 参数指定的 UDP 端口块中的 UDP 端口，则返回此错误。如果由 <i>StartPort</i> 和 <i>NumberOfPorts</i> 参数指定的 UDP 端口块的永久性预留与已创建的 UDP 端口块的永久性预留匹配或重叠，也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

备注

CreatePersistentUdpPortReservation 函数在 Windows Vista 及更高版本上定义。

CreatePersistentUdpPortReservation 函数用于为 UDP 端口块添加永久性预留。

需要保留端口的应用程序和服务分为两类。第一个类别包括需要特定端口作为其操作的一部分的组件。此类组件通常倾向于在应用程序清单中（在安装时指定所需的端口，例如）。第二个类别包括在运行时需要任何可用端口或端口块的组件。

这两个类别对应于特定和通配符端口预留请求。特定预留请求可以是永久性的或运行时的，而通配符端口预留请求仅在运行时受支持。

CreatePersistentUdpPortReservation 函数使应用程序或服务能够永久保留 UDP 端口块。永久性 TCP 预留记录在 Windows 中 UDP 模块的持久存储中。

调用方通过指定所需的端口数以及是否需要特定范围来获取永久性端口预留。如果可以满足请求，**CreatePersistentUdpPortReservation** 函数将返回唯一不透明的 ULONG64 令牌，该令牌随后标识预留。可以通过调用 [DeletePersistentUdpPortReservation](#) 函数释放永久性 UDP 端口预留。请注意，每次重启系统时，给定的持久 UDP 端口预留的令牌可能会更改。

Windows 不会为使用这些函数获取的永久性预留实现组件间安全性。这意味着，如果向某个组件授予获取任何永久性端口预留的能力，该组件将自动获得使用授予系统上任何其他组件的任何永久性端口预留的能力。对运行时预留强制实施进程级安全性，但此类控制不能扩展到使用 [CreatePersistentTcpPortReservation](#) 或 [CreatePersistentUdpPortReservation](#) 函数创建的创建的永久性预留。

获取持久 UDP 端口预留后，应用程序可以通过打开 UDP 套接字，然后调用 [WSAIoctl](#) 函数（指定 [SIO_ASSOCIATE_PORT_RESERVATION](#) IOCTL 并在调用套接字上的 [绑定](#) 函数之

前传递预留令牌) 从 UDP 端口预留请求端口分配。

[SIO_ACQUIRE_PORT_RESERVATION](#) IOCTL 可用于请求 TCP 或 UDP 端口块的运行时预留。对于运行时端口预留，端口池要求从向其授予预留的套接字的进程使用预留。运行时端口预留的持续时间仅为调用 [SIO_ACQUIRE_PORT_RESERVATION](#) IOCTL 的套接字的生存期。相比之下，使用 [CreatePersistentUdpPortReservation](#) 函数创建的永久性端口预留可由能够获取永久性预留的任何进程使用。

[CreatePersistentUdpPortReservation](#) 函数只能由以 Administrators 组成员身份登录的用户调用。如果不是 Administrators 组成员的用户调用

[CreatePersistentUdpPortReservation](#)，则函数调用将失败并返回

[ERROR_ACCESS_DENIED](#)。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 [requestedExecutionLevel](#) 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[CreatePersistentTcpPortReservation](#)

[DeletePersistentTcpPortReservation](#)

[DeletePersistentUdpPortReservation](#)

[LookupPersistentTcpPortReservation](#)

[LookupPersistentUdpPortReservation](#)

SIO_ACQUIRE_PORT_RESERVATION

SIO_ASSOCIATE_PORT_RESERVATION

SIO_RELEASE_PORT_RESERVATION

WSAIoctl

bind

反馈

此页面是否有帮助?

 是

 否

在 Microsoft Q&A 获得帮助

CreateProxyArpEntry 函数 (iphlpapi.h)

项目2023/08/24

CreateProxyArpEntry 函数在本地计算机上为指定的 IPv4 地址创建代理地址解析协议 (PARP) 条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD CreateProxyArpEntry(
    [in] DWORD dwAddress,
    [in] DWORD dwMask,
    [in] DWORD dwIfIndex
);
```

parameters

[in] dwAddress

此计算机充当代理的 IPv4 地址。

[in] dwMask

dwAddress 中指定的 IPv4 地址的子网掩码。

[in] dwIfIndex

要为其代理 *dwAddress* 标识的 IPv4 地址的 ARP 的接口的索引。 换句话说，在此接口上收到 *dwAddress* 的 ARP 请求时，本地计算机会使用此接口的物理地址进行响应。 如果此接口的类型不支持 ARP（如 PPP），则调用将失败。

返回值

如果函数成功，则函数返回 NO_ERROR（零）。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的

	管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员（运行方式管理员）。
ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。如果 <i>dwAddress</i> 参数为零或无效值，其他参数之一无效，则返回此错误。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

若要检索 ARP 表，请调用 [GetIpNetTable](#) 函数。若要删除现有的 P ARP 条目，请调用 [DeleteProxyArpEntry](#)。

在 Windows Vista 及更高版本上，[CreateProxyArpEntry](#) 函数只能由以管理员组成员身份登录的用户调用。如果 [CreateProxyArpEntry](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 [ERROR_ACCESS_DENIED](#)。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 [requestedExecutionLevel](#) 设置为 requireAdministrator。如果 Windows Vista 和更高版本上的应用程序缺少此清单文件，则以管理员组成员身份登录的用户（而不是内置管理员）必须在增强的 shell 中执行应用程序，因为内置管理员（RunAs 管理员）才能使此功能成功。

注意 此函数执行特权操作。若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[DeleteProxyArpEntry](#)

[GetIpNetTable](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_PROXYARP](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

CreateSortedAddressPairs 函数 (netioapi.h)

项目2023/08/25

`CreateSortedAddressPairs` 函数采用提供的潜在 IP 目标地址列表，将目标地址与主机的本地 IP 地址配对，并根据最适合两个对等之间的通信的地址对进行排序。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
CreateSortedAddressPairs(
    [in, optional] const PSOCKADDR_IN6 SourceAddressList,
    [in]           ULONG             SourceAddressCount,
    [in]           const PSOCKADDR_IN6 DestinationAddressList,
    [in]           ULONG             DestinationAddressCount,
    [in]           ULONG             AddressSortOptions,
    [out]          PSOCKADDR_IN6_PAIR *SortedAddressPairList,
    [out]          ULONG             *SortedAddressPairCount
);
```

parameters

[in, optional] `SourceAddressList`

必须为 `NULL`。保留供将来使用。

[in] `SourceAddressCount`

必须为 0。保留供将来使用。

[in] `DestinationAddressList`

指向包含潜在 IPv6 目标地址列表 的`SOCKADDR_IN6` 结构的数组的指针。任何 IPv4 地址都必须以 IPv4 映射的 IPv6 地址格式表示，这允许仅 IPv6 应用程序与 IPv4 节点通信。

[in] `DestinationAddressCount`

`DestinationAddressList` 参数指向的目标地址数。

[in] `AddressSortOptions`

保留供将来使用。

[out] `SortedAddressPairList`

一个指针，用于存储 `SOCKADDR_IN6_PAIR` 结构的数组，这些结构包含按首选通信顺序排序的 IPv6 地址对列表（如果函数调用成功）。

[out] `SortedAddressPairCount`

一个指针，用于存储 `SortedAddressPairList` 参数指向的地址对数（如果函数调用成功）。

返回值

如果函数成功，则返回值 `NO_ERROR`。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
<code>ERROR_INVALID_PARAMETER</code>	向该函数传递了无效参数。如果 <code>DestinationAddressList</code> 、 <code>SortedAddressPairList</code> 或 <code>SortedAddressPairCount</code> 参数 <code>NULL</code> 或 <code>DestinationAddressCount</code> 大于 500，则返回此错误。如果 <code>SourceAddressList</code> 不是 <code>NULL</code> 或 <code>SourceAddressPairCount</code> 参数不为零，也会返回此错误。
<code>ERROR_NOT_ENOUGH_MEMORY</code>	没有足够的存储空间可用于处理此命令。
<code>ERROR_NOT_SUPPORTED</code>	不支持该请求。如果本地计算机上没有 IPv6 堆栈，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`CreateSortedAddressPairs` 函数是在 Windows Vista 及更高版本上定义的。

`CreateSortedAddressPairs` 函数采用源和目标 IPv6 地址的列表，并按排序顺序返回地址对列表。列表按哪个地址对最适合源地址和目标地址之间的通信进行排序。

`SourceAddressList` 指向的源地址列表目前保留供将来使用，并且必须是 `NULL` 指针。

`SourceAddressCount` 目前保留供将来使用，并且必须为零。`CreateSortedAddressPairs` 函数当前将主机的所有本地地址用于源地址列表。

目标地址列表由 `DestinationAddressList` 参数指向。目标地址列表是 `SOCKADDR_IN6` 结构的数组。任何 IPv4 地址都必须以 IPv4 映射的 IPv6 地址格式表示，这允许仅 IPv6 应用

程序与 IPv4 节点通信。有关 IPv4 映射的 IPv6 地址格式的详细信息，请参阅 [双堆栈套接字](#)。*DestinationAddressCount* 参数包含 *DestinationAddressList* 参数指向的目标地址数。*CreateSortedAddressPairs* 函数最多支持 500 个目标地址。

如果 *CreateSortedAddressPairs* 函数成功，则 *SortedAddressPairList* 参数指向包含已排序地址对的 [SOCKADDR_IN6_PAIR](#) 结构的数组。如果不再需要此返回的列表，请通过调用 [FreeMibTable](#) 函数释放列表使用的内存。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[双堆栈套接字](#)

[FreeMibTable](#)

[SOCKADDR_IN6_PAIR](#)

[使用 SIO_ADDRESS_LIST_SORT](#)

[sockaddr](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

CreateUnicastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

CreateUnicastIpAddressEntry 函数在本地计算机上添加新的单播 IP 地址条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
CreateUnicastIpAddressEntry(
    [in] const MIB_UNICASTIPADDRESS_ROW *Row
);
```

parameters

[in] Row

指向单播 IP 地址条目 [MIB_UNICASTIPADDRESS_ROW](#) 结构条目的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数指向 MIB_UNICASTIPADDRESS_ROW 的 Address 成员未设置为有效的单播 IPv4 或 IPv6 地址，或者未指定 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW 的 InterfaceLuid 和 InterfaceIndex 成员，则返回此错误。 对于为 MIB_UNICASTIPADDRESS_ROW 结构中的成员设置的值中的其他错误，也会返回此错误。这些错误包括以下

	错误：如果 <code>ValidLifetime</code> 成员小于 <code>PreferredLifetime</code> 成员，如果 <code>PrefixOrigin</code> 成员设置为 <code>IpPrefixOriginUnchanged</code> ， <code>SuffixOrigin</code> 未设置为 <code>IpSuffixOriginUnchanged</code> ，如果 <code>PrefixOrigin</code> 成员未设置为 <code>IpPrefixOriginUnchanged</code> ，并且 <code>SuffixOrigin</code> 设置为 <code>IpSuffixOriginUnchanged</code> ，如果 <code>PrefixOrigin</code> 如果 <code>SuffixOrigin</code> 成员未设置为 <code>NL_SUFFIX_ORIGIN</code> 枚举中的值，或者如果 <code>OnLinkPrefixLength</code> 成员设置为大于 IP 地址长度的值（以位（32 表示单播 IPv4 地址）或 128（对于单播 IPv6 地址）），则 <code>member</code> 不会设置为来自 <code>NL_PREFIX_ORIGIN</code> 枚举的值。
<code>ERROR_NOT_FOUND</code>	找不到指定的接口。如果找不到由 <code>Row</code> 参数指向的 MIB_UNICASTIPADDRESS_ROW 的 <code>InterfaceLuid</code> 或 <code>InterfaceIndex</code> 成员指定的网络接口，则返回此错误。
<code>ERROR_NOT_SUPPORTED</code>	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且已在 <code>Row</code> 参数指向的 MIB_UNICASTIPADDRESS_ROW 的 <code>Address</code> 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，则也会返回此错误。
<code>ERROR_OBJECT_ALREADY_EXISTS</code>	该对象已经存在。如果 <code>Row</code> 参数指向的 MIB_UNICASTIPADDRESS_ROW 的 <code>Address</code> 成员是 MIB_UNICASTIPADDRESS_ROW 的 <code>InterfaceLuid</code> 或 <code>InterfaceIndex</code> 成员指定的接口上现有单播 IP 地址的副本，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`CreateUnicastIpAddressEntry` 函数是在 Windows Vista 及更高版本上定义的。

`CreateUnicastIpAddressEntry` 函数用于在本地计算机上添加新的单播 IP 地址条目。由 `CreateUnicastIpAddressEntry` 函数添加的单播 IP 地址不是永久性的。只要适配器对象存在，IP 地址才存在。重新启动计算机会破坏 IP 地址，手动重置网络接口卡 (NIC)。此外，某些 PnP 事件可能会销毁地址。

若要创建持久化 IPv4 地址，可以使用 Windows Management Instrumentation (WMI) 控件中 [Win32_NetworkAdapterConfiguration](#) 类的 [EnableStatic](#) 方法。netsh 命令还可用于创建永久性 IPv4 或 IPv6 地址。

有关详细信息，请参阅 Windows 套接字文档中有关 [Netsh.exe](#) 的文档。

`InitializeUnicastIpAddressEntry` 函数应用于使用默认值初始化 [MIB_UNICASTIPADDRESS_ROW](#) 结构条目的成员。然后，应用程序可以更改要修改的

`MIB_UNICASTIPADDRESS_ROW` 条目中的成员，然后调用 `CreateUnicastIpAddressEntry` 函数。

`Row` 参数指向的 `MIB_UNICASTIPADDRESS_ROW` 结构中的 `Address` 成员必须初始化为有效的单播 IPv4 或 IPv6 地址。 `Address` 成员中 `SOCKADDR_INET` 结构的 `si_family` 成员必须初始化为 `AF_INET` 或 `AF_INET6`，并且必须将 `SOCKADDR_INET` 结构的相关 `Ipv4` 或 `Ipv6` 成员设置为有效的单播 IP 地址。此外，必须将指向 `Row` 参数的 `MIB_UNICASTIPADDRESS_ROW` 结构中的以下成员中的至少一个初始化到接口：`InterfaceLuid` 或 `InterfaceIndex`。

字段按上面列出的顺序使用。因此，如果指定了 `InterfaceLuid`，则此成员用于确定要添加单播 IP 地址的接口。如果没有为 `InterfaceLuid` 成员设置值，(此成员的值) 设置为零，则接下来使用 `InterfaceIndex` 成员来确定接口。

如果 `Row` 参数指向的 `MIB_UNICASTIPADDRESS_ROW` 的 `OnLinkPrefixLength` 成员设置为 255，则 `CreateUnicastIpAddressEntry` 将添加新的单播 IP 地址，其 `OnLinkPrefixLength` 成员集等于 IP 地址的长度。因此，对于单播 IPv4 地址，对于单播 IPv6 地址，`OnLinkPrefixLength` 设置为 32，`OnLinkPrefixLength` 设置为 128。如果这会导致 IPv4 地址的子网掩码不正确或 IPv6 地址的链接前缀不正确，则应用程序应在调用 `CreateUnicastIpAddressEntry` 之前将此成员设置为正确的值。

如果在未正确设置 `OnLinkPrefixLength` 成员的情况下创建单播 IP 地址，则可以通过调用 `SetUnicastIpAddressEntry` 并将 `OnLinkPrefixLength` 成员设置为正确值来更改 IP 地址。

调用 `CreateUnicastIpAddressEntry` 函数时，将忽略行指向的 `MIB_UNICASTIPADDRESS_ROW` 结构的 `DadState`、`ScopId` 和 `CreationTimeStamp` 成员。这些成员由网络堆栈设置。`ScopId` 成员由添加地址的接口自动确定。从 Windows 10 开始，如果在调用 `CreateUnicastIpAddressEntry` 时将 `dadState` 设置为 `MIB_UNICASTIPADDRESS_ROW` 结构中的 `IpDadStatePreferred`，则堆栈会将地址的初始 DAD 状态设置为“首选”而不是“暂定”，并将对地址执行乐观 DAD。

如果 `Row` 参数指向的 `MIB_UNICASTIPADDRESS_ROW` 的 `Address` 成员中传递的单播 IP 地址是接口上现有单播 IP 地址的副本，则 `CreateUnicastIpAddressEntry` 函数将失败。请注意，环回 IP 地址只能使用 `CreateUnicastIpAddressEntry` 函数添加到环回接口。

`Row` 参数指向的 `MIB_UNICASTIPADDRESS_ROW` 的 `Address` 成员中传递的单播 IP 地址不能立即使用。重复地址检测过程成功完成后，IP 地址可用。重复地址检测过程可能需要几秒钟才能完成，因为需要发送 IP 数据包，并且必须等待潜在的响应。对于 IPv6，重复地址检测过程通常需要大约一秒钟的时间。对于 IPv4，重复地址检测过程通常需要大约三秒钟。

如果应用程序在调用 `CreateUnicastIpAddressEntry` 函数后需要知道 IP 地址何时可用，则可以使用两种方法。一种方法使用轮询和 `GetUnicastIpAddressEntry` 函数。第二种方

法调用通知函数之一 [NotifyAddrChange](#)、[NotifyIpInterfaceChange](#) 或 [NotifyUnicastIpAddressChange](#)，以在地址更改时设置异步通知。

以下方法介绍如何使用 [GetUnicastIpAddressEntry](#) 和轮询。成功返回对 [CreateUnicastIpAddressEntry](#) 函数的调用后，暂停一到三秒（具体取决于是创建 IPv6 还是 IPv4 地址），以便有时间成功完成重复地址检测过程。然后调用 [GetUnicastIpAddressEntry](#) 函数以检索更新 [MIB_UNICASTIPADDRESS_ROW](#) 结构并检查 [DadState](#) 成员的值。如果 [DadState](#) 成员的值设置为 [IpDadStatePreferred](#)，则 IP 地址现在可用。如果 [DadState](#) 成员的值设置为 [IpDadStateTentative](#)，则重复地址检测尚未完成。在这种情况下，每半秒调用一次 [GetUnicastIpAddressEntry](#) 函数，而 [DadState](#) 成员仍设置为 [IpDadStateTentative](#)。如果 [DadState](#) 成员的值返回的某个值不是 [IpDadStatePreferred](#) 或 [IpDadStateTentative](#)，则重复地址检测失败，IP 地址不可用。

以下方法介绍如何使用适当的通知函数。成功返回对 [CreateUnicastIpAddressEntry](#) 函数的调用后，调用 [NotifyUnicastIpAddressChange](#) 函数以注册以收到 IPv6 或 IPv4 单播 IP 地址更改的通知，具体取决于要创建的 IP 地址的类型。收到要创建的 IP 地址的通知时，调用 [GetUnicastIpAddressEntry](#) 函数以检索 [DadState](#) 成员。如果 [DadState](#) 成员的值设置为 [IpDadStatePreferred](#)，则 IP 地址现在可用。如果 [DadState](#) 成员的值设置为 [IpDadStateTentative](#)，则重复地址检测尚未完成，应用程序需要等待将来的通知。如果 [DadState](#) 成员的值返回的某个值不是 [IpDadStatePreferred](#) 或 [IpDadStateTentative](#)，则重复地址检测失败，IP 地址不可用。

如果在重复地址检测过程中，媒体断开连接，然后重新连接，则重复地址检测进程会重启。因此，在完成此过程时，可能会增加超过 IPv6 的典型 1 秒值或 IPv4 的 3 秒值。

[CreateUnicastIpAddressEntry](#) 函数只能由以管理员组成员身份登录的用户调用。如果 [CreateUnicastIpAddressEntry](#) 由非管理员组成员的用户调用，则函数调用将失败并返回 [ERROR_ACCESS_DENIED](#)。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 [requestedExecutionLevel](#) 设置为 [requireAdministrator](#)。如果上的应用程序缺少此清单文件，则以管理员组成员身份登录的用户（而不是内置管理员）必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员)，此函数才能成功。

示例

以下示例演示如何使用 [CreateUnicastIpAddressEntry](#) 函数在本地计算机上添加新的单播 IP 地址条目。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
```

```
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>
#include <winsock2.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

// Need to link with Iphlpapi.lib and Ws2_32.lib
#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

HANDLE gCallbackComplete;
HANDLE gNotifyEvent;

void CALLBACK CallCompleted (VOID *callerContext,
    PMIB_UNICASTIPADDRESS_ROW row,
    MIB_NOTIFICATION_TYPE notificationType);

int main(int argc, char **argv) {

    // Declare and initialize variables

    unsigned long ipAddress = INADDR_NONE;
    unsigned long ipMask = INADDR_NONE;

    DWORD dwRetVal = 0;

    DWORD dwSize = 0;
    unsigned long status = 0;

    DWORD lastError = 0;
    SOCKADDR_IN localAddress;

    NET_LUID interfaceLuid;
    PMIB_IPINTERFACE_TABLE pipTable = NULL;
    MIB_UNICASTIPADDRESS_ROW ipRow;

    // Validate the parameters
    if (argc != 3) {
        printf("usage: %s IPv4address IPv4mask\n", argv[0]);
        exit(1);
    }

    ipAddress = inet_addr(argv[1]);
    if (ipAddress == INADDR_NONE) {
        printf("usage: %s IPv4address IPv4mask\n", argv[0]);
        exit(1);
    }

    ipMask = inet_addr(argv[2]);
    if (ipMask == INADDR_NONE) {
        printf("usage: %s IPv4address IPv4mask\n", argv[0]);
    }
}
```

```

    exit(1);
}

status = GetIpInterfaceTable( AF_INET, &pipTable );
if( status != NO_ERROR )
{
    printf("GetIpInterfaceTable returned error: %ld\n",
          status);
    exit(1);
}

// Use loopback interface
interfaceLuid = pipTable->Table[0].InterfaceLuid;

localAddress.sin_family      = AF_INET;
localAddress.sin_addr.S_un.S_addr = ipAddress;

FreeMibTable(pipTable);
pipTable = NULL;

// Initialize the row
InitializeUnicastIpAddressEntry( &ipRow );

ipRow.InterfaceLuid = interfaceLuid;
ipRow.Address.Ipv4 = localAddress;

// Create a Handle to be notified of IP address changes
gCallbackComplete = CreateEvent(NULL, FALSE, FALSE, NULL);
if (gCallbackComplete == NULL) {
    printf("CreateEvent failed with error: %d\n", GetLastError() );
    exit(1);
}

// Use NotifyUnicastIpAddressChange to determine when the address is
ready
NotifyUnicastIpAddressChange(AF_INET, &CallCompleted, NULL, FALSE,
&gNotifyEvent);

status = CreateUnicastIpAddressEntry(&ipRow);
if(status != NO_ERROR)
{
    CancelMibChangeNotify2(gNotifyEvent);
    switch(status)
    {
        case ERROR_INVALID_PARAMETER:
            printf("Error: CreateUnicastIpAddressEntry returned
ERROR_INVALID_PARAMETER\n");
            break;
        case ERROR_NOT_FOUND:
            printf("Error: CreateUnicastIpAddressEntry returned
ERROR_NOT_FOUND\n");
            break;
        case ERROR_NOT_SUPPORTED:
            printf("Error: CreateUnicastIpAddressEntry returned

```

```

        ERROR_NOT_SUPPORTED\n");
        break;
    case ERROR_OBJECT_ALREADY_EXISTS:
        printf("Error: CreateUnicastIpAddressEntry returned
ERROR_OBJECT_ALREADY_EXISTS\n");
        break;
    default:
        //NOTE: Is this case needed? If not, we can remove the
        ErrorExit() function
        printf("CreateUnicastIpAddressEntry returned error: %d\n",
status);
        break;
    }
    exit (status);

}

else
    printf("CreateUnicastIpAddressEntry succeeded\n");

// Set timeout to 6 seconds
status = WaitForSingleObject(gCallbackComplete, 6000);
if(status != WAIT_OBJECT_0)
{
    CancelMibChangeNotify2(gNotifyEvent);
    CancelMibChangeNotify2(gCallbackComplete);
    switch(status)
    {
        case WAIT_ABANDONED:
            printf("Wait on event was abandoned\n");
            break;
        case WAIT_TIMEOUT:
            printf("Wait on event timed out\n");
            break;
        default:
            printf("Wait on event exited with status %d\n", status);
            break;
    }
    return status;
}
printf("Task completed successfully\n");
CancelMibChangeNotify2(gNotifyEvent);
CancelMibChangeNotify2(gCallbackComplete);

exit (0);
}

void CALLBACK CallCompleted(PVOID callerContext, PMIB_UNICASTIPADDRESS_ROW
row, MIB_NOTIFICATION_TYPE notificationType)
{
    ADDRESS_FAMILY addressFamily;
    SOCKADDR_IN sockv4addr;
    struct in_addr ipv4addr;
}

```

```

    // Ensure that this is the correct notification before setting
    gCallbackComplete
    // NOTE: Is there a stronger way to do this?
    if(notificationType == MibAddInstance) {
        printf("NotifyUnicastIpAddressChange received an Add instance\n");
        addressFamily = (ADDRESS_FAMILY) row->Address.si_family;
        switch (addressFamily) {
            case AF_INET:
                printf("\tAddressFamily: AF_INET\n");
                break;
            case AF_INET6:
                printf("\tAddressFamily: AF_INET6\n");
                break;
            default:
                printf("\tAddressFamily: %d\n", addressFamily);
                break;
        }
        if (addressFamily == AF_INET) {
            sockv4addr = row->Address.Ipv4;
            ipv4addr = sockv4addr.sin_addr;
            printf("IPv4 address: %s\n", inet_ntoa(ipv4addr) );
        }
        if (callerContext != NULL)
            printf("Received a CallerContext value\n");

        SetEvent(gCallbackComplete);
    }
    return;
}

```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[DeleteUnicastIpAddressEntry](#)

[GetUnicastIpAddressEntry](#)

[GetUnicastIpAddressTable](#)

IP 帮助程序函数参考

[InitializeUnicastIpAddressEntry](#)

[MIB_UNICASTIPADDRESS_ROW](#)

[MIB_UNICASTIPADDRESS_TABLE](#)

[Netsh.exe](#)

[NotifyAddrChange](#)

[NotifyIpInterfaceChange](#)

[NotifyUnicastIpAddressChange](#)

[SetUnicastIpAddressEntry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

deleteAnycastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

DeleteAnycastIpAddressEntry 函数删除本地计算机上的现有 anycast IP 地址条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
DeleteAnycastIpAddressEntry(
    [in] const MIB_ANYCASTIPADDRESS_ROW *Row
);
```

parameters

[in] Row

指向要从本地计算机中删除的现有 anycast IP 地址条目的 MIB_ANYCASTIPADDRESS_ROW 结构条目的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置的管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数所指向 MIB_ANYCASTIPADDRESS_ROW 的 Address 成员未设置为有效的单播 IPv4 或 IPv6 地址，或者未指定 Row 参数所指向 MIB_ANYCASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误。

ERROR_NOT_FOUND	找不到指定的接口。如果找不到 <i>Row</i> 参数指向的 MIB_ANYCASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 <i>Row</i> 参数指向的 Address 成员 MIB_ANYCASTIPADDRESS_ROW 指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`DeleteAnycastIpAddressEntry` 函数在 Windows Vista 及更高版本上定义。

`DeleteAnycastIpAddressEntry` 函数用于删除本地计算机上的现有 [MIB_ANYCASTIPADDRESS_ROW](#) 结构条目。

输入时，*Row* 参数指向 [MIB_ANYCASTIPADDRESS_ROW](#) 结构中的 **Address** 成员必须设置为有效的单播 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 *Row* 参数的 [MIB_ANYCASTIPADDRESS_ROW](#) 结构中的至少一个成员：**InterfaceLuid** 或 **InterfaceIndex**。

字段按上面列出的顺序使用。因此，如果指定了 **InterfaceLuid**，则使用此成员来确定接口。如果未为 **InterfaceLuid** 成员设置值（此成员的值设置为零），则接下来使用 **InterfaceIndex** 成员来确定接口。

如果函数成功，则已删除 *Row* 参数表示的现有 IP 地址。

可以调用 [GetAnycastIpAddressTable](#) 函数来枚举本地计算机上的 anycast IP 地址条目。可以调用 [GetAnycastIpAddressEntry](#) 函数来检索特定的现有 anycast IP 地址条目。

`DeleteAnycastIpAddressEntry` 函数只能由以 Administrators 组成员身份登录的用户调用。如果 `DeleteAnycastIpAddressEntry` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 **ERROR_ACCESS_DENIED**。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 **requestedExecutionLevel** 设置为 **requireAdministrator**。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateAnycastIpAddressEntry](#)

[GetAnycastIpAddressEntry](#)

[GetAnycastIpAddressTable](#)

[IP 帮助程序函数参考](#)

[MIB_ANYCASTIPADDRESS_ROW](#)

[MIB_ANYCASTIPADDRESS_TABLE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

deleteIPAddress 函数 (iphlpapi.h)

项目2023/08/24

DeleteIPAddress 函数删除以前使用 [AddIPAddress 添加的](#) IP 地址。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD DeleteIPAddress(
    [in] ULONG NTEContext
);
```

parameters

[in] NTEContext

Net Table 条目 (NTE) IP 地址的上下文。此上下文是由对 [AddIPAddress 的](#)上一次调用返回的。

返回值

如果函数成功，则函数返回 NO_ERROR (零)。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (运行方式管理员)。
ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

在 Windows Vista 及更高版本中，只能由以管理员组成员身份登录的用户调用 DeleteIPAddress 函数。如果 DeleteIPAddress 由非管理员组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果 Windows Vista 和更高版本上的应用程序缺少此清单文件，则以管理员组成员身份登录的用户（而不是内置管理员）必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

示例

以下示例检索 IP 地址表，然后将 IP 地址 192.168.0.27 添加到第一个适配器。然后删除添加的 IP 地址。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

int main()
{
    // Declare and initialize variables
    PMIB_IPADDRTABLE pIPAddrTable;
    DWORD dwSize = 0;
    DWORD dwRetVal;

    // IP and mask we will be adding
    UINT iaIPAddress;
    UINT imIPMask;
```

```

// Variables where handles to the added IP will be returned
ULONG NTEContext = 0;
ULONG NTEInstance = 0;

LPVOID lpMsgBuf;

// Before calling AddIPAddress we use GetIpAddrTable to get
// an adapter to which we can add the IP.
pIPAddrTable = (MIB_IPADDRTABLE *) malloc(sizeof(MIB_IPADDRTABLE));

// Make an initial call to GetIpAddrTable to get the
// necessary size into the dwSize variable
if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==
ERROR_INSUFFICIENT_BUFFER) {
    GlobalFree(pIPAddrTable);
    pIPAddrTable = (MIB_IPADDRTABLE *) malloc(dwSize);
}

// Make a second call to GetIpAddrTable to get the
// actual data we want
if ((dwRetVal = GetIpAddrTable(pIPAddrTable, &dwSize, 0)) == NO_ERROR) {
    printf("\tAddress: %ld\n", pIPAddrTable->table[0].dwAddr);
    printf("\tMask:    %ld\n", pIPAddrTable->table[0].dwMask);
    printf("\tIndex:   %ld\n", pIPAddrTable->table[0].dwIndex);
    printf("\tBCast:   %ld\n", pIPAddrTable->table[0].dwBCastAddr);
    printf("\tReasm:   %ld\n", pIPAddrTable->table[0].dwReasmSize);
} else {
    printf("Call to GetIpAddrTable failed.\n");
}

// IP and mask we will be adding

iaIPAddress = inet_addr("192.168.0.27");
imIPMask = inet_addr("255.255.255.0");

if ((dwRetVal = AddIPAddress(iaIPAddress,
                            imIPMask,
                            pIPAddrTable->table[0].dwIndex,
                            &NTEContext, &NTEInstance)) == NO_ERROR) {
    printf("\tIP address added.\n");
}

else {
    printf("Error adding IP address.\n");

    if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL, dwRetVal,
MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),           // Default language
(LPTSTR) & lpMsgBuf, 0, NULL)) {
        printf("\tError: %s", lpMsgBuf);
    }
    LocalFree(lpMsgBuf);
}

// Delete the IP we just added using the NTEContext

```

```
// variable where the handle was returned
if ((dwRetVal = DeleteIPAddress(NTEContext)) == NO_ERROR) {
    printf("\tIP Address Deleted.\n");
} else {
    printf("\tCall to DeleteIPAddress failed.\n");
}

exit(0);
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[AddIPAddress](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

deleteIpForwardEntry 函数 (iphlpapi.h)

项目2023/08/24

DeleteIpForwardEntry 函数删除本地计算机的 IPv4 路由表中的现有路由。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD DeleteIpForwardEntry(
    [in] PMIB_IPFORWARDROW pRoute
);
```

parameters

[in] pRoute

指向 MIB_IPFORWARDROW 结构的指针。此结构指定用于标识要删除的路由的信息。调用方必须为结构的 dwForwardIfIndex、dwForwardDest、dwForwardMask、dwForwardNextHop 和 dwForwardProto 成员指定值。

返回值

如果例程成功，函数返回NO_ERROR(零)。

如果函数失败，则返回值为以下错误代码之一。

错误代码	含义
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员(运行方式管理员)。
ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。如果 pRoute 参数为 NULL、PMIB_IPFORWARDROW 结构的 dwForwardMask 成员不是有效的 IPv4 子网掩码、dwForwardIfIndex 成员为 NULL，或者 MIB_IPFORWARDROW 结构的其他成员之一无效，则返回此错误。
ERROR_NOT_FOUND	pRoute 参数指向不存在的路由条目。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。

(其他)

该函数可能会返回其他错误代码。

如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

注解

必须将 *route* 参数 [MIB_IPFORWARDROW](#) 结构指针的 *dwForwardProto* 成员设置为 [MIB IPPROTO_NETMGMT](#) 否则 [DeleteIpForwardEntry](#) 将失败。 路由协议标识符用于标识指定路由协议的路由信息。 例如， [MIB IPPROTO_NETMGMT](#) 用于通过网络管理（例如动态主机配置协议 (DHCP)、简单网络管理协议 (SNMP)）或通过调用 [CreateIpForwardEntry](#)、[DeleteIpForwardEntry](#) 或 [SetIpForwardEntry](#) 函数来标识 IP 路由设置的路由信息。

在 Windows Vista 和 Windows Server 2008 上，[DeleteIpForwardEntry](#) 仅适用于具有单个子接口的接口（其中接口 LUID 和子接口 LUID 是相同的）。 [MIB_IPFORWARDROW](#) 结构的 *dwForwardIfIndex* 成员指定接口。

[CreateIpForwardEntry](#) 当前不使用路由参数指向的 [MIB_IPFORWARDROW](#) 结构中的许多成员。 这些成员包括 *dwForwardPolicy*、*dwForwardType*、*dwForwardAge*、*dwForwardNextHopAS*、*dwForwardMetric1*、*dwForwardMetric2*、*dwForwardMetric3*、*dwForwardMetric4* 和 *dwForwardMetric5*。

若要修改 IPv4 路由表中的现有路由，请使用 [SetIpForwardEntry](#) 函数。 若要检索 IPv4 路由表，请调用 [GetIpForwardTable](#) 函数。

在 Windows Vista 及更高版本上，只能由以管理员组成员身份登录的用户调用 [DeleteIpForwardEntry](#) 函数。 如果 [DeleteIpForwardEntry](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 [ERROR_ACCESS_DENIED](#)。

[DeleteIpForwardEntry](#) 函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。 如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 *requestedExecutionLevel* 设置为 *requireAdministrator*。 如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。 若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

示例

下面的代码示例演示如何将默认网关更改为 NewGateway。通过调用 GetIpForwardTable，更改网关，然后调用 SetIpForwardEntry 不会更改路由，但会添加新路由。如果存在多个默认网关，此代码将删除它们。请注意，新网关必须可行;否则，TCP/IP 将忽略更改。

注意 执行此代码将更改 IP 路由表，并可能导致网络活动失败。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "iphlpapi.lib")

int main()
{
    // Declare and initialize variables
    PMIB_IPFORWARDTABLE pIpForwardTable = NULL;
    PMIB_IPFORWARDROW pRow = NULL;
    DWORD dwSize = 0;
    BOOL bOrder = FALSE;
    DWORD dwStatus = 0;
    DWORD NewGateway = 0xDDDBCCAA;           // this is in host order Ip Address
    AA.BB.CC.DD is DDCCBAA

    unsigned int i;

    // Identify the required size of the buffer.
    dwStatus = GetIpForwardTable(pIpForwardTable, &dwSize, bOrder);
    if (dwStatus == ERROR_INSUFFICIENT_BUFFER) {
        // Allocate memory for the table.
        if (!(pIpForwardTable = (PMIB_IPFORWARDTABLE) malloc(dwSize))) {
            printf("Malloc failed. Out of memory.\n");
            exit(1);
        }
        // Retrieve the table.
        dwStatus = GetIpForwardTable(pIpForwardTable, &dwSize, bOrder);
    }

    if (dwStatus != ERROR_SUCCESS) {
```

```

        printf("getIpForwardTable failed.\n");
        if (pIpForwardTable)
            free(pIpForwardTable);
        exit(1);
    }
// Search for the required row in the table. The default gateway has a
destination
// of 0.0.0.0. Be aware the table continues to be searched, but only
// one row is copied. This is to ensure that, if multiple gateways exist,
all of them are deleted.
//
for (i = 0; i < pIpForwardTable->dwNumEntries; i++) {
    if (pIpForwardTable->table[i].dwForwardDest == 0) {
        // The default gateway was found.
        if (!pRow) {
            // Allocate memory to store the row. This is easier than
manually filling
            // the row structure; only the gateway address is changed.
            //
            pRow = (PMIB_IPFORWARDROW) malloc(sizeof
(MIB_IPFORWARDROW));
            if (!pRow) {
                printf("Malloc failed. Out of memory.\n");
                exit(1);
            }
            // Copy the row.
            memcpy(pRow, &(pIpForwardTable->table[i]),
sizeof (MIB_IPFORWARDROW));
        }
        // Delete the old default gateway entry.
        dwStatus = DeleteIpForwardEntry(&(pIpForwardTable->table[i]));

        if (dwStatus != ERROR_SUCCESS) {
            printf("Could not delete old gateway\n");
            exit(1);
        }
    }
}

// Set the nexthop field to our new gateway. All other properties of the
route will
// remain the same.
pRow->dwForwardNextHop = NewGateway;

// Create a new route entry for the default gateway.
dwStatus = CreateIpForwardEntry(pRow);

if (dwStatus == NO_ERROR)
    printf("Gateway changed successfully\n");
else if (dwStatus == ERROR_INVALID_PARAMETER)
    printf("Invalid parameter.\n");
else
    printf("Error: %d\n", dwStatus);

// Free the memory.

```

```
    if (pIpForwardTable)
        free(pIpForwardTable);
    if (pRow)
        free(pRow);

    exit(0);
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateIpForwardEntry](#)

[FormatMessage](#)

[GetIpForwardTable](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPFORWARDROW](#)

[SetIpForwardEntry](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

deleteIpForwardEntry2 函数 (netioapi.h)

项目2023/08/25

DeleteIpForwardEntry2 函数删除本地计算机上的 IP 路由条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API DeleteIpForwardEntry2(
    [in] const MIB_IPFORWARD_ROW2 *Row
);
```

parameters

[in] Row

指向 IP 路由条目 [MIB_IPFORWARD_ROW2](#) 结构条目的指针。 成功返回后，此条目将被删除。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。 此错误在以下几种情况下返回： 用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置的管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果在 Row 参数中传递 NULL 指针，未指定 Row 参数所指向 MIB_IPFORWARD_ROW2 的 DestinationPrefix 成员、Row 参数指向 MIB_IPFORWARD_ROW2 的 NextHop 成员，或者 Row 参数指向的 MIB_IPFORWARD_ROW2 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误未指定参数。
ERROR_NOT_FOUND	找不到指定的接口。 如果找不到 Row 参数指向的 MIB_IPFORWARD_ROW2 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口，则返回此错误。

ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Row 参数指向的 MIB_IPFORWARD_ROW2 的 Address 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`DeleteIpForwardEntry2` 函数在 Windows Vista 及更高版本上定义。

`DeleteIpForwardEntry2` 函数用于删除 MIB_IPFORWARD_ROW2 结构条目。

输入时，Row 参数指向的 MIB_IPFORWARD_ROW2 结构中的 DestinationPrefix 成员必须初始化为有效的 IPv4 或 IPv6 地址前缀和系列。输入时，Row 参数指向的 MIB_IPFORWARD_ROW2 结构中的 NextHop 成员必须初始化为有效的 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 Row 参数的 MIB_IPFORWARD_ROW2 结构中的至少一个成员：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则使用此成员来确定接口。如果未为 InterfaceLuid 成员设置值（此成员的值设置为零），则接下来使用 InterfaceIndex 成员来确定接口。

在调用成功时输出时，`DeleteIpForwardEntry2` 将删除 IP 路由条目。

如果 Row 参数指向的 MIB_IPFORWARD_ROW2 的 DestinationPrefix 和 NextHop 成员与 InterfaceLuid 或 InterfaceIndex 成员中指定的接口上的现有 IP 路由条目不匹配，`DeleteIpForwardEntry2` 函数将失败。

可以调用 `GetIpForwardTable2` 函数来枚举本地计算机上的 IP 路由条目。

`DeleteIpForwardEntry2` 函数只能由以 Administrators 组成员身份登录的用户调用。如果 `DeleteIpForwardEntry2` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpForwardEntry2](#)

[GetBestRoute2](#)

[GetIpForwardEntry2](#)

[GetIpForwardTable2](#)

[InitializeIpForwardEntry](#)

[MIB_IPFORWARD_ROW2](#)

[MIB_IPFORWARD_TABLE2](#)

[NotifyRouteChange2](#)

[SetIpForwardEntry2](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

deleteIpNetEntry 函数 (iphlpapi.h)

项目2023/08/24

DeleteIpNetEntry 函数从本地计算机上的 ARP 表中删除 ARP 条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD DeleteIpNetEntry(
    [in] PMIB_IPNETROW pArpEntry
);
```

parameters

[in] pArpEntry

指向 [MIB_IPNETROW](#) 结构的指针。此结构中的信息指定要删除的条目。调用方必须至少为此结构的 dwIndex 和 dwAddr 成员指定值。

返回值

如果函数成功，则函数返回 NO_ERROR (零)。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (运行方式管理员)。
ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。如果 pArpEntry 参数为 NULL 或 pArpEntry 参数指向的 MIB_IPNETROW 结构中的某个成员无效，则返回此错误。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

若要检索 ARP 表，请调用 [GetIpNetTable](#) 函数。

在 Windows Vista 及更高版本上，[DeleteIpNetEntry](#) 函数只能由以管理员组成员身份登录的用户调用。如果 [DeleteIpNetEntry](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 [ERROR_ACCESS_DENIED](#)。

[DeleteIpNetEntry](#) 函数也可能因为用户帐户控制 (Windows Vista 及更高版本上的 UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 [requestedExecutionLevel](#) 设置为 [requireAdministrator](#)。如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateIpNetEntry](#)

[FlushIpNetTable](#)

[GetIpNetTable](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPNETROW](#)

[SetIpNetEntry](#)

反馈

此页面是否有帮助？

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

deleteIpNetEntry2 函数 (netioapi.h)

项目2023/08/25

DeleteIpNetEntry2 函数删除本地计算机上的相邻 IP 地址条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API DeleteIpNetEntry2(
    [in] const MIB_IPNET_ROW2 *Row
);
```

parameters

[in] Row

指向邻居 IP 地址条目 [MIB_IPNET_ROW2](#) 结构条目的指针。 成功返回后，此条目将被删除。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。 此错误在以下几种情况下返回： 用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置的管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果在 Row 参数中传递 NULL 指针，Row 参数指向 MIB_IPNET_ROW2 的 Address 成员未设置为有效的邻居 IPv4 或 IPv6 地址，或者未指定 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误。
ERROR_NOT_FOUND	找不到指定的接口。 如果找不到 Row 参数指向的 MIB_IPNET_ROW2 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口，则返回此错误。

ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Row 参数指向的 MIB_IPNET_ROW2 的 Address 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`DeleteIpNetEntry2` 函数在 Windows Vista 及更高版本上定义。

`DeleteIpNetEntry2` 函数用于删除 MIB_IPNET_ROW2 结构条目。

输入时，Row 参数指向的 MIB_IPNET_ROW2 结构中的 Address 成员必须初始化为有效的邻居 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 Row 参数的 MIB_IPNET_ROW2 结构中的至少一个成员：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则使用此成员来确定接口。如果未为 InterfaceLuid 成员设置值（此成员的值设置为零），则接下来使用 InterfaceIndex 成员来确定接口。

当调用成功时输出时，`DeleteIpNetEntry2` 会删除邻居 IP 地址。

可以调用 [GetIpNetTable2](#) 函数来枚举本地计算机上的邻居 IP 地址条目。

`DeleteIpNetEntry2` 函数只能由以 Administrators 组成员身份登录的用户调用。如果 `DeleteIpNetEntry2` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows

标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpNetEntry2](#)

[FlushIpNetTable2](#)

[GetIpNetEntry2](#)

[GetIpNetTable2](#)

[MIB_IPNET_ROW2](#)

[MIB_IPNET_TABLE2](#)

[ResolveIpNetEntry2](#)

[SetIpNetEntry2](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

DeletePersistentTcpPortReservation 函数 (iphlpapi.h)

项目2023/08/24

DeletePersistentTcpPortReservation 函数删除本地计算机上连续 TCP 端口块的永久性 TCP 端口预留。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG DeletePersistentTcpPortReservation(
    [in] USHORT StartPort,
    [in] USHORT NumberOfPorts
);
```

parameters

[in] StartPort

按网络字节顺序排列的起始 TCP 端口号。

[in] NumberOfPorts

要删除的 TCP 端口号数。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 StartPort 或 NumberOfPorts 参数中传递零，则返回此错误。

ERROR_NOT_FOUND	找不到该元素。如果找不到由 <i>StartPort</i> 和 <i>NumberOfPorts</i> 参数指定的永久性端口块，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

DeletePersistentTcpPortReservation 函数在 Windows Vista 及更高版本上定义。

DeletePersistentTcpPortReservation 函数用于删除 TCP 端口块的永久性预留。

DeletePersistentTcpPortReservation 函数只能由以管理员组成员身份登录的用户调用。如果 **DeletePersistentTcpPortReservation** 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 **ERROR_ACCESS_DENIED**。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 **requestedExecutionLevel** 设置为 **requireAdministrator**。如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

示例

以下示例删除永久性 TCP 端口预留。

此示例必须由管理员组成员的用户运行。运行此示例的最简单方法是使用增强的 shell，因为内置管理员 (RunAs 管理员)。

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <Windows.h>
#include <winsock2.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

// Need to link with iphlpapi.lib
```

```
#pragma comment(lib, "iphlpapi.lib")

// Need to link with ws2_32.lib for htons
#pragma comment(lib, "ws2_32.lib")

int wmain(int argc, WCHAR **argv) {

    // Declare and initialize variables

    int startPort = 0;           // host byte order
    int numPorts = 0;
    USHORT startPortns = 0;      // Network byte order

    unsigned long status = 0;

    // Validate the parameters
    if (argc != 3) {
        wprintf(L"usage: %s <Starting Port> <Number of Ports>\n", argv[0]);
        wprintf(L"Delete a persistent TCP port reservation\n");
        wprintf(L"Example usage:\n");
        wprintf(L"    %s 5000 20\n", argv[0]);
        wprintf(L"    where StartPort=5000 NumPorts=20");
        return 1;
    }

    startPort = _wtoi(argv[1]);
    if ( startPort < 0 || startPort > 65535) {
        wprintf(L"Starting point must be either 0 or between 1 and
65,535\n");
        return 1;
    }
    startPortns = htons((u_short) startPort);

    numPorts = _wtoi(argv[2]);
    if (numPorts < 0) {
        wprintf(L"Number of ports must be a positive number\n");
        return 1;
    }

    status = DeletePersistentTcpPortReservation((USHORT) startPortns,
(USHORT) numPorts);
    if( status != NO_ERROR )
    {
        wprintf(L"DeletePersistentTcpPortReservation returned error: %ld\n",
status);
        return 1;
    }

    wprintf(L"DeletePersistentTcpPortReservation call succeeded\n");

    return 0;
}
```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreatePersistentTcpPortReservation](#)

[CreatePersistentUdpPortReservation](#)

[DeletePersistentUdpPortReservation](#)

[LookupPersistentTcpPortReservation](#)

[LookupPersistentUdpPortReservation](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

deletePersistentUdpPortReservation 函数 (iphlpapi.h)

项目2023/08/24

DeletePersistentUdpPortReservation 函数删除本地计算机上 TCP 端口连续块的永久性 TCP 端口预留。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG DeletePersistentUdpPortReservation(
    [in] USHORT StartPort,
    [in] USHORT NumberOfPorts
);
```

parameters

[in] StartPort

按网络字节顺序排列的起始 UDP 端口号。

[in] NumberOfPorts

要删除的 UDP 端口号数。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置的管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 StartPort 或 NumberOfPorts 参数中传递零，则返回此错误。

ERROR_NOT_FOUND	找不到该元素。如果找不到由 <i>StartPort</i> 和 <i>NumberOfPorts</i> 参数指定的永久性端口块，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

DeletePersistentUdpPortReservation 函数在 Windows Vista 及更高版本上定义。

DeletePersistentUdpPortReservation 函数用于删除 UDP 端口块的永久性预留。

DeletePersistentUdpPortReservation 函数只能由以 Administrators 组成员身份登录的用户调用。如果 **DeletePersistentUdpPortReservation** 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 **ERROR_ACCESS_DENIED**。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 **requestedExecutionLevel** 设置为 **requireAdministrator**。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[CreatePersistentTcpPortReservation](#)

[CreatePersistentUdpPortReservation](#)

[DeletePersistentTcpPortReservation](#)

[LookupPersistentTcpPortReservation](#)

[LookupPersistentUdpPortReservation](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

deleteProxyArpEntry 函数 (iphlpapi.h)

项目2023/08/24

DeleteProxyArpEntry 函数删除由 *dwAddress* 和 *dwIfIndex* 参数指定的本地计算机上的 P ARP 条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD DeleteProxyArpEntry(
    [in] DWORD dwAddress,
    [in] DWORD dwMask,
    [in] DWORD dwIfIndex
);
```

parameters

[in] *dwAddress*

此计算机充当代理的 IPv4 地址。

[in] *dwMask*

dwAddress 参数中指定的 IPv4 地址的子网掩码。

[in] *dwIfIndex*

此计算机支持 *dwAddress* 参数指定的 IP 地址的代理 ARP 的接口的索引。

返回值

如果函数成功，则函数返回 NO_ERROR (零)。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (运行方式管理员)。

ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

若要检索 ARP 表，请调用 [GetIpNetTable](#) 函数。

在 Windows Vista 及更高版本上，`DeleteProxyArpEntry` 函数只能由以管理员组成员身份登录的用户调用。如果 `DeleteProxyArpEntry` 由非管理员组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果 Windows Vista 和更高版本上的应用程序缺少此清单文件，则以管理员组成员身份登录的用户（而不是内置管理员）必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

注意 此函数执行特权操作。若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[CreateProxyArpEntry](#)

[GetIpNetTable](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_PROXYARP](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

deleteUnicastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

DeleteUnicastIpAddressEntry 函数删除本地计算机上的现有单播 IP 地址条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
DeleteUnicastIpAddressEntry(
    [in] const MIB_UNICASTIPADDRESS_ROW *Row
);
```

parameters

[in] Row

指向要从本地计算机中删除的现有单播 IP 地址条目 的MIB_UNICASTIPADDRESS_ROW 结构条目的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置的管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数所指向MIB_UNICASTIPADDRESS_ROW的 Address 成员未设置为有效的单播 IPv4 或 IPv6 地址，或者未指定 Row 参数指向的 MIB_UNICASTIPADDRESS_ROWInterfaceLuid 或 InterfaceIndex 成员，则返回此错误。

ERROR_NOT_FOUND	找不到指定的接口。如果找不到 <i>Row</i> 参数指向的 MIB_UNICASTIPADDRESS_ROW <i>InterfaceLuid</i> 或 <i>InterfaceIndex</i> 成员指定的网络接口，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 <i>Row</i> 参数指向的 <i>Address</i> 成员 MIB_UNICASTIPADDRESS_ROW 指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[DeleteUnicastIpAddressEntry](#) 函数在 Windows Vista 及更高版本上定义。

[DeleteUnicastIpAddressEntry](#) 函数用于删除本地计算机上的现有 [MIB_UNICASTIPADDRESS_ROW](#) 结构条目。

输入时，*Row* 参数指向 [MIB_UNICASTIPADDRESS_ROW](#) 结构中的 *Address* 成员必须设置为有效的单播 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 *Row* 参数的 [MIB_UNICASTIPADDRESS_ROW](#) 结构中的至少一个成员：*InterfaceLuid* 或 *InterfaceIndex*。

字段按上面列出的顺序使用。因此，如果指定了 *InterfaceLuid*，则使用此成员来确定接口。如果未为 *InterfaceLuid* 成员设置值（此成员的值设置为零），则接下来使用 *InterfaceIndex* 成员来确定接口。

如果函数成功，则已删除 *Row* 参数表示的现有 IP 地址。

可以调用 [GetUnicastIpAddressTable](#) 函数来枚举本地计算机上的单播 IP 地址条目。可以调用 [GetUnicastIpAddressEntry](#) 函数来检索特定的现有单播 IP 地址条目。

[DeleteUnicastIpAddressEntry](#) 函数只能由以 Administrators 组成员身份登录的用户调用。如果 [DeleteUnicastIpAddressEntry](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 [ERROR_ACCESS_DENIED](#)。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 **requestedExecutionLevel** 设置为 **requireAdministrator**。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateUnicastIpAddressEntry](#)

[GetUnicastIpAddressEntry](#)

[GetUnicastIpAddressTable](#)

[IP 帮助程序函数参考](#)

[InitializeUnicastIpAddressEntry](#)

[MIB_UNICASTIPADDRESS_ROW](#)

[MIB_UNICASTIPADDRESS_TABLE](#)

[NotifyUnicastIpAddressChange](#)

[SetUnicastIpAddressEntry](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

DisableMediaSense 函数 (iphlpapi.h)

项目2023/08/24

DisableMediaSense 函数禁用本地计算机上的 TCP/IP 堆栈的媒体感知功能。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD DisableMediaSense(
    HANDLE      *pHandle,
    OVERLAPPED *pOverLapped
);
```

parameters

pHandle

指向用于存储句柄的变量的指针。如果 *pOverlapped* 参数不为 NULL，则此变量将在内部用于存储调用 IP 驱动程序和禁用媒体感知功能所需的句柄。

应用程序不应使用此变量指向的值。此句柄供内部使用，不应关闭。

pOverLapped

指向 OVERLAPPED 结构的指针。除 hEvent 成员外，此结构的所有成员都必须设置为零。hEvent 成员需要有效事件对象的句柄。使用 [CreateEvent](#) 函数创建此事件对象。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>pOverlapped</i> 参数是错误的指针，则返回此错误。
ERROR_IO_PENDING	操作正在进行中。此值由对 DisableMediaSense 的成功异步调用返回。

ERROR_OPEN_FAILED	<i>pHandle</i> 参数指向的句柄无效。
ERROR_NOT_SUPPORTED	不支持该请求。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

如果 *pHandle* 或 *pOverlapped* 参数为 NULL，则 [DisableMediaSense](#) 函数将同步执行。

如果 *pHandle* 和 *pOverlapped* 参数都不为 NULL，则使用 *pOverlapped* 参数指向的 [OVERLAPPED](#) 结构异步执行 [DisableMediaSense](#) 函数。

在稍后调用 [RestoreMediaSense](#) 函数以还原媒体感知功能之前，[DisableMediaSense](#) 函数不会完成。在此之前，I/O 请求数据包 (IRP) 仍排队。或者，当名为 [DisableMediaSense](#) 的进程退出时，将取消 IRP，并调用取消例程，以再次还原媒体感知功能。

若要以同步方式调用 [DisableMediaSense](#)，应用程序需要为此调用创建单独的线程。否则，它会继续等待 IRP 完成，函数将阻止。

若要异步调用 [DisableMediaSense](#)，应用程序需要分配一个 [OVERLAPPED](#) 结构。除 *hEvent* 成员外，此结构的所有成员都必须设置为零。*hEvent* 成员需要有效事件对象的句柄。使用 [CreateEvent](#) 函数创建此事件。异步调用 [DisableMediaSense](#) 时，始终返回 [ERROR_IO_PENDING](#)。仅当稍后调用 [RestoreMediaSense](#) 时，才会完成 IRP。当不再需要事件对象时，使用 [CloseHandle](#) 函数关闭该事件对象的句柄。进程终止时，系统会自动关闭句柄。事件对象在关闭其最后一个句柄时被销毁。

在 Windows Server 2003 和 Windows XP 上，TCP/IP 堆栈实现删除接口上所有 IP 地址的策略，以响应与基础网络接口的媒体感知断开连接事件。如果本地计算机连接到的网络交换机或集线器已关闭电源，或者网络电缆断开连接，则网络接口将传递断开连接事件。与网络接口关联的 IP 配置信息丢失。因此，TCP/IP 堆栈实现隐藏断开连接的接口的策略，以便这些接口及其关联的 IP 地址不会显示在通过 IP 帮助程序检索的配置信息中。此策略可防止某些应用程序轻松检测到网络接口只是断开连接，而不是从系统中删除。

如果本地客户端计算机使用 DHCP 请求获取 IP 配置信息的 DHCP 服务器，则此行为通常不会影响本地客户端计算机。但这会对服务器计算机（尤其是用作群集一部分的计算机）产生严重影响。[DisableMediaSense](#) 函数可用于暂时禁用这些情况下的媒体感知功能。稍后会调用 [RestoreMediaSense](#) 函数来还原媒体感知功能。

以下注册表设置与 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数相关：

系统\CurrentControlSet\服务\Tcpip\参数\DisableDHCPMediaSense

如果计算机首次启动时此注册表项存在，则 Windows 中会设置一个内部标志。通过调用 [DisableMediaSense](#) 和 [RestoreMediaSense](#)，还可以设置和重置同一内部标志。但是，使用注册表设置时，需要重新启动计算机才能发生更改。

Windows Vista 和更高版本的 TCP/IP 堆栈已更改为在发生断开连接事件时不隐藏断开连接的接口。因此，在 Windows Vista 及更高版本中，[DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数不会执行任何操作，并且始终返回NO_ERROR。

示例

以下示例演示如何异步调用 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数。此示例仅适用于 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数执行有用操作的 Windows Server 2003 和 Windows XP。

该示例首先调用 [DisableMediaSense](#) 函数，休眠 60 秒以允许用户断开网络电缆的连接，检索 IP 地址表并打印表中的 IP 地址条目的某些成员，调用 [RestoreMediaSense](#) 函数，再次检索 IP 地址表，并打印表中的一些 IP 地址条目的成员。禁用媒体感知功能的影响可以从 IP 地址表条目的差异中看出。

有关演示如何同步调用 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数的示例，请参阅 [RestoreMediaSense](#) 函数参考。

C++

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int __cdecl main()
{
    int i;

    /* Variables used by GetIpAddrTable */
    PMIB_IPADDRTABLE pIPAddrTable;
    DWORD dwSize = 0;
    DWORD dwRetVal = 0;
    IN_ADDR IPAddr;
```

```

/* Variables used to return error message */
LPVOID lpMsgBuf;

// Variables to call DisableMediaSense
// and RestoreMediaSense asynchronously
HANDLE IpDriverHandle = INVALID_HANDLE_VALUE;
OVERLAPPED Overlapped;
HANDLE DriverHandle;
DWORD dwEnableCount = 0;

memset(&Overlapped, 0, sizeof(Overlapped));
Overlapped.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

dwRetVal = DisableMediaSense(&DriverHandle, &Overlapped);
if (dwRetVal != ERROR_IO_PENDING) {
    printf("DisableMediaSense failed with error %d\n", dwRetVal);
    exit(1);
} else {
    printf(" === DisableMediaSense called ===\n\n");
    // Sleep for 60 seconds so we can disconnect a cable
    Sleep(60000);
}

// Before calling AddIPAddress we use GetIpAddrTable to get
// an adapter to which we can add the IP.
pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(sizeof(MIB_IPADDRTABLE));

if (pIPAddrTable) {
    // Make an initial call to GetIpAddrTable to get the
    // necessary size into the dwSize variable
    if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==
        ERROR_INSUFFICIENT_BUFFER) {
        FREE(pIPAddrTable);
        pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(dwSize);

    }
    if (pIPAddrTable == NULL) {
        printf("Memory allocation failed for GetIpAddrTable\n");
        exit(1);
    }
}
// Make a second call to GetIpAddrTable to get the
// actual data we want
if ((dwRetVal = GetIpAddrTable(pIPAddrTable, &dwSize, 0)) != NO_ERROR) {
    printf("GetIpAddrTable failed with error %d\n", dwRetVal);
    if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                      FORMAT_MESSAGE_FROM_SYSTEM |
                      FORMAT_MESSAGE_IGNORE_INSERTS,
                      NULL,
                      dwRetVal,
                      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default
language
                      (LPTSTR) & lpMsgBuf, 0, NULL)) {
        printf("\tError: %s", lpMsgBuf);
        LocalFree(lpMsgBuf);
    }
}

```

```

    }

    exit(1);
}

printf("\tNum Entries: %ld\n", pIPAddrTable->dwNumEntries);
for (i = 0; i < (int) pIPAddrTable->dwNumEntries; i++) {
    printf("\n\tInterface Index[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwIndex);
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwAddr;
    printf("\tIP Address[%d]: \t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwMask;
    printf("\tSubnet Mask[%d]: \t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwBcastAddr;
    printf("\tBroadCast[%d]: \t%s (%ld)\n", i, inet_ntoa(IPAddr),
           pIPAddrTable->table[i].dwBcastAddr);
    printf("\tReassembly size[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwReasmSize);
    printf("\tType and State[%d]:", i);
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_PRIMARY)
        printf("\tPrimary IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DYNAMIC)
        printf("\tDynamic IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DISCONNECTED)
        printf("\tAddress is on disconnected interface");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DELETED)
        printf("\tAddress is being deleted");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_TRANSIENT)
        printf("\tTransient address");
    printf("\n");
}

// Call RestoreMediaSense asynchronously to enable mediasense
dwRetVal = RestoreMediaSense(&Overlapped, &dwEnableCount);
if (dwRetVal && dwRetVal != ERROR_IO_PENDING) {
    printf("RestoreMediaSense failed with error %d\n", dwRetVal);
    exit(1);
} else {
    printf(" === RestoreMediaSense called ===\n");
    printf("  EnableCount returned was %ld\n\n", dwEnableCount);
}

if (pIPAddrTable) {
    // Make an initial call to GetIpAddrTable to get the
    // necessary size into the dwSize variable
    if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==
        ERROR_INSUFFICIENT_BUFFER) {
        FREE(pIPAddrTable);
        pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(dwSize);

    }
    if (pIPAddrTable == NULL) {
        printf("Memory allocation failed for GetIpAddrTable\n");
        exit(1);
    }
}

```

```

// Make a second call to GetIpAddrTable to get the
// actual data we want
if ((dwRetVal = GetIpAddrTable(pIPAddrTable, &dwSize, 0)) != NO_ERROR) {
    printf("GetIpAddrTable failed with error %d\n", dwRetVal);
    if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                      FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
                      NULL, dwRetVal,
                      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default
language
                      (LPTSTR) & lpMsgBuf, 0, NULL)) {
        printf("\tError: %s", lpMsgBuf);
        LocalFree(lpMsgBuf);
    }
    exit(1);
}

printf("\tNum Entries: %ld\n", pIPAddrTable->dwNumEntries);
for (i = 0; i < (int) pIPAddrTable->dwNumEntries; i++) {
    printf("\n\tInterface Index[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwIndex);
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwAddr;
    printf("\tIP Address[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwMask;
    printf("\tSubnet Mask[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwBcastAddr;
    printf("\tBroadCast[%d]:\t%s (%ld)\n", i, inet_ntoa(IPAddr),
           pIPAddrTable->table[i].dwBcastAddr);
    printf("\tReassembly size[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwReasmSize);
    printf("\tType and State[%d]:", i);
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_PRIMARY)
        printf("\tPrimary IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DYNAMIC)
        printf("\tDynamic IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DISCONNECTED)
        printf("\tAddress is on disconnected interface");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DELETED)
        printf("\tAddress is being deleted");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_TRANSIENT)
        printf("\tTransient address");
    printf("\n");
}

if (pIPAddrTable) {
    FREE(pIPAddrTable);
    pIPAddrTable = NULL;
}

exit(0);
}

```

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CloseHandle](#)

[CreateEvent](#)

[EnableRouter](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[OVERLAPPED](#)

[RestoreMediaSense](#)

[UnenableRouter](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

enableRouter 函数 (iphlpapi.h)

项目2023/08/24

EnableRouter 函数在本地计算机上打开 IPv4 转发。 EnableRouter 还会递增引用计数，用于跟踪启用 IPv4 转发的请求数。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD EnableRouter(
    HANDLE      *pHandle,
    OVERLAPPED *pOverlapped
);
```

parameters

pHandle

指向句柄的指针。 此参数当前未使用。

pOverlapped

指向 OVERLAPPED 结构的指针。 除 hEvent 成员外，此结构的所有成员都应设置为零。 hEvent 成员应包含有效事件对象的句柄。 使用 [CreateEvent](#) 函数创建此事件对象。

返回值

如果 EnableRouter 函数成功，则返回值ERROR_IO_PENDING。

如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

返回代码	说明
ERROR_INVALID_PARAMETER	其中一个参数无效。 如果 <i>pOverlapped</i> 参数为 NULL，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`EnableRouter` 函数特定于 IPv4 转发。如果调用 `EnableRouter` 的进程在未调用 `UnenableRouter` 的情况下终止，系统会减少跟踪启用 IPv4 转发的请求数的引用计数，就像进程调用了 `UnenableRouter` 一样。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateEvent](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[OVERLAPPED](#)

[UnenableRouter](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

FlushIpNetTable 函数 (iphlpapi.h)

项目2023/08/24

FlushIpNetTable 函数从本地计算机上的 ARP 表中删除指定接口的所有 ARP 条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD FlushIpNetTable(
    [in] DWORD dwIfIndex
);
```

parameters

[in] dwIfIndex

要删除其所有 ARP 条目的接口的索引。

返回值

如果函数成功，则函数返回 NO_ERROR (零)。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (运行方式管理员)。
ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。如果 dwIfIndex 参数无效，则返回此错误。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

若要检索 ARP 表，请调用 [GetIpNetTable](#) 函数。

在 Windows Vista 及更高版本上，只有以管理员组成员身份登录的用户才能调用 `FlushIpNetTable` 函数。如果 `FlushIpNetTable` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果 Windows Vista 和更高版本上的应用程序缺少此清单文件，则以管理员组成员身份登录的用户（而不是内置管理员）必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

注意 此函数执行特权操作。若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateIpNetEntry](#)

[DeleteIpNetEntry](#)

[GetIfTable](#)

[GetIpNetTable](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

FlushIpNetTable2 函数 (netioapi.h)

项目2023/08/25

FlushIpNetTable2 函数刷新本地计算机上的 IP 邻居表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API FlushIpNetTable2(
    [in] ADDRESS_FAMILY Family,
    [in] NET_IFINDEX      InterfaceIndex
);
```

parameters

[in] Family

要刷新的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和 PF_ 协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 *Ws2def.h* 头文件中定义了此成员的可能值。请注意，*Ws2def.h* 头文件会自动包含在 *Winsock2.h* 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	地址系列未指定。指定此参数后，此函数将刷新包含 IPv4 和 IPv6 条目的邻居 IP 地址表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数时，此函数将刷新仅包含 IPv4 条目的邻居 IP 地址表。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数后，此函数将刷新仅包含 IPv6 条目的邻居 IP 地址表。

[in] InterfaceIndex

接口索引。如果指定了索引，请刷新特定接口上的邻居 IP 地址条目，否则刷新所有接口上的邻居 IP 地址条目。若要忽略 接口，请将此参数设置为零。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果未将 Family 参数指定为 AF_INET、AF_INET6 或 AF_UNSPEC，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且已在 Family 参数中指定了 AF_INET，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且已在 Family 参数中指定了 AF_INET6，则也会返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`FlushIpNetTable2` 函数在 Windows Vista 及更高版本上定义。

The

`FlushIpNetTable2` 函数刷新或删除本地系统上的相邻 IP 地址。*Family* 参数可用于将要删除的邻居 IP 地址限制为特定 IP 地址系列。如果应删除 IPv4 和 IPv6 的邻居 IP 地址，请将 *Family* 参数设置为 AF_UNSPEC。*InterfaceIndex* 参数可用于限制要删除到特定接口的邻居 IP 地址。如果应删除所有接口的邻居 IP 地址，请将 *InterfaceIndex* 参数设置为零。

Family 参数必须初始化为 AF_INET、AF_INET6 或 AF_UNSPEC。

`FlushIpNetTable2` 函数只能由以管理员组成员身份登录的用户调用。如果 `FlushIpNetTable2` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文

件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员（RunAs 管理员）才能使此功能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpNetEntry2](#)

[DeleteIpNetEntry2](#)

[GetIpNetEntry2](#)

[GetIpNetTable2](#)

[MIB_IPNET_ROW2](#)

[MIB_IPNET_TABLE2](#)

[ResolveIpNetEntry2](#)

[SetIpNetEntry2](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

FlushIpPathTable 函数 (netioapi.h)

项目2023/08/25

FlushIpPathTable 函数刷新本地计算机上的 IP 路径表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API FlushIpPathTable(  
    [in] ADDRESS_FAMILY Family  
) ;
```

parameters

[in] Family

要刷新的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和PF_ 协议系列常量的值 (相同，例如，AF_INET 和 PF_INET) ，因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 *Ws2def.h* 头文件中定义了此成员的可能值。请注意，*Ws2def.h* 头文件会自动包含在 *Winsock2.h* 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	地址系列未指定。指定此参数后，此函数将刷新包含 IPv4 和 IPv6 条目的 IP 路径表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数时，此函数将刷新仅包含 IPv4 条目的 IP 路径表。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数后，此函数将刷新仅包含 IPv6 条目的 IP 路径表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果未将 <i>Family</i> 参数指定为 AF_INET、AF_INET6 或 AF_UNSPEC，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且已在 <i>Family</i> 参数中指定了 AF_INET，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且已在 <i>Family</i> 参数中指定了 AF_INET6，则也会返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

FlushIpPathTable 函数在 Windows Vista 及更高版本上定义。

The

FlushIpPathTable 函数刷新或删除本地系统上的 IP 路径条目。*Family* 参数可用于将要删除的 IP 路径条目限制为特定的 IP 地址系列。如果应删除 IPv4 和 IPv6 的 IP 路径条目，请将 *Family* 参数设置为 AF_UNSPEC。

Family 参数必须初始化为 AF_INET、AF_INET6 或 AF_UNSPEC。

FlushIpPathTable 函数只能由以 Administrators 组成员身份登录的用户调用。如果 **FlushIpPathTable** 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

要求

最低受支持的客户端

Windows Vista [仅限桌面应用]

最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[GetIpPathEntry](#)

[GetIpPathTable](#)

[MIB_IPPATH_ROW](#)

[MIB_IPPATH_TABLE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

FreeInterfaceDnsSettings 函数 (netioapi.h)

项目2023/04/22

释放 [GetInterfaceDnsSettings 返回的设置对象](#)。

语法

C++

```
VOID NETIOAPI_API_ FreeInterfaceDnsSettings(
    DNS_INTERFACE_SETTINGS *Settings
);
```

parameters

Settings

要求

最低受支持的客户端	Windows 10内部版本 19041
最低受支持的服务器	Windows 10内部版本 19041
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

- [GetInterfaceDnsSettings](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

freeMibTable 函数 (netioapi.h)

项目2023/08/25

FreeMibTable 函数释放函数分配的缓冲区，这些函数 ([GetIfTable2](#) 和 [GetAnycastIpAddressTable](#) 返回网络接口、地址和路由表，例如）。

语法

C++

```
IPHLPAPI_DLL_LINKAGE VOID NETIOAPI_API_ FreeMibTable(
    [in] PVOID Memory
);
```

parameters

[in] Memory

指向要释放的缓冲区的指针。

返回值

此函数不返回值。

注解

FreeMibTable 函数在 Windows Vista 及更高版本上定义。

FreeMibTable 函数用于释放各种函数用于检索接口、地址和路由表的内部缓冲区。如果不再需要这些表，则应调用 **FreeMibTable** 以释放这些表使用的内存。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows

标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[GetAnycastIpAddressTable](#)

[GetIfStackTable](#)

[GetIfTable2](#)

[GetIfTable2Ex](#)

[GetInvertedIfStackTable](#)

[GetIpForwardTable2](#)

[GetIpInterfaceTable](#)

[GetIpNetTable2](#)

[GetIpPathTable](#)

[GetMulticastIpAddressTable](#)

[GetUnicastIpAddressTable](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

GetAdapterIndex 函数 (iphlpapi.h)

项目2023/08/24

GetAdapterIndex 函数根据名称获取适配器的索引。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetAdapterIndex(
    [in]      LPWSTR AdapterName,
    [in, out] PULONG IfIndex
);
```

parameters

[in] AdapterName

指向 Unicode 字符串的指针，该字符串指定适配器的名称。

[in, out] IfIndex

指向指向适配器索引的 ULONG 变量的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

注解

在完全禁用适配器之前，GetAdapterIndex 函数将适配器报告为存在。例如，[NotifyAddrChange](#) 函数可能指示删除了最近禁用的适配器的 IP 地址，但 GetAdapterIndex 会继续报告适配器索引，直到禁用适配器的过程完成。

当系统上存在一个或多个适配器时，当查询的适配器不存在时，GetAdapterIndex 将返回ERROR_DEV_NOT_EXIST。如果没有适配器，GetAdapterIndex 函数将返回ERROR_NO_DATA。

禁用并启用适配器时，或者在其他情况下，适配器索引可能会更改，不应被视为永久性。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetAdaptersInfo](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IP_ADAPTER_INFO](#)

[MprConfigGetFriendlyName](#)

[MprConfigGetGuidName](#)

[NotifyAddrChange](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

GetAdapterOrderMap 函数 (iphlpapi.h)

项目2023/08/24

GetAdapterOrderMap 函数获取适配器顺序映射，该映射指示本地计算机上的接口的优先级。

语法

C++

```
IPHLPAPI_DLL_LINKAGE PIP_ADAPTER_ORDER_MAP GetAdapterOrderMap();
```

返回值

返回一个 [IP_ADAPTER_ORDER_MAP](#) 结构，其中填充了适配器优先级信息。有关详细信息，请参阅 [IP_ADAPTER_ORDER_MAP](#) 结构。

注解

接口索引按“高级设置”属性表中的“适配器和绑定”对话框中指定的顺序显示。此排序用作连接断路器，用于控制接口在多宿主系统上用于路由选择、DNS 名称解析和其他网络相关操作的顺序。

不应直接调用此函数。请改用 [GetAdaptersInfo](#) 函数调用中返回的 [IP_ADAPTER_INFO](#) 结构。

注意 调用方负责调用 [LocalFree](#) 函数以释放 [GetAdapterOrderMap](#) 返回的数组。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows

标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetAdaptersInfo](#)

[IP_ADAPTER_INFO](#)

[IP_ADAPTER_ORDER_MAP](#)

[LocalFree](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

GetAdaptersAddresses 函数 (iphlpapi.h)

项目2023/08/24

GetAdaptersAddresses 函数检索与本地计算机上的适配器关联的地址。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetAdaptersAddresses(
    [in]      ULONG           Family,
    [in]      ULONG           Flags,
    [in]      PVOID           Reserved,
    [in, out] PIP_ADAPTER_ADDRESSES AdapterAddresses,
    [in, out] PULONG          SizePointer
);
```

parameters

[in] Family

要检索的地址的地址系列。此参数须为下列值之一。

值	含义
AF_UNSPEC 0	返回与启用了 IPv4 或 IPv6 的适配器关联的 IPv4 和 IPv6 地址。
AF_INET 2	仅返回与启用了 IPv4 的适配器关联的 IPv4 地址。
AF_INET6 23	仅返回与启用了 IPv6 的适配器关联的 IPv6 地址。

[in] Flags

要检索的地址类型。可能的值在 *Iptypes.h* 头文件中定义。请注意，*Iptypes.h* 头文件会自动包含在 *Iphlpapi.h* 中，永远不应直接使用。

此参数是以下值的组合。如果此参数为零，则将返回单播、任意广播和多播 IP 地址。

值	含义
GAA_FLAG_SKIP_UNICAST	不要返回单播地址。

0x0001	
GAA_FLAG_SKIP_ANycast 0x0002	不要返回 IPv6 任何广播地址。
GAA_FLAG_SKIP_MULTICAST 0x0004	不要返回多播地址。
GAA_FLAG_SKIP_DNS_SERVER 0x0008	不要返回 DNS 服务器的地址。
GAA_FLAG_INCLUDE_PREFIX 0x0010	返回此适配器上的 IP 地址前缀列表。 设置此标志后，将返回 IPv6 和 IPv4 地址的 IP 地址前缀。 具有 SP1 和更高版本的 Windows XP 支持此标志。
GAA_FLAG_SKIP_FRIENDLY_NAME 0x0020	不要返回适配器友好名称。
GAA_FLAG_INCLUDE_WINS_INFO 0x0040	返回 Windows Internet 名称服务的地址 (WINS) 服务器。 Windows Vista 及更高版本支持此标志。
GAA_FLAG_INCLUDE_GATEWAYS 0x0080	返回默认网关的地址。 Windows Vista 及更高版本支持此标志。
GAA_FLAG_INCLUDE_ALL_INTERFACES 0x0100	返回所有 NDIS 接口的地址。 Windows Vista 及更高版本支持此标志。
GAA_FLAG_INCLUDE_ALL_COMPARTMENTS 0x0200	返回所有路由隔离舱中的地址。 此标志当前不受支持，并保留供将来使用。
GAA_FLAG_INCLUDE_TUNNEL_BINDINGORDER 0x0400	返回按隧道绑定顺序排序的适配器地址。 Windows Vista 及更高版本支持此标志。

[in] Reserved

此参数当前未使用，但保留供将来系统使用。 调用应用程序应为此参数传递 **NULL**。

[in, out] AdapterAddresses

指向缓冲区的指针，该缓冲区包含成功返回时 [IP_ADAPTER_ADDRESSES](#) 结构的链接列表。

[in, out] SizePointer

指向变量的指针，该变量指定 *AdapterAddresses* 指向的缓冲区的大小。

返回值

如果函数成功，则返回值 `ERROR_SUCCESS` (定义为与 `NO_ERROR`) 相同的值。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
<code>ERROR_ADDRESS_NOT_ASSOCIATED</code>	地址尚未与网络终结点关联。DHCP 租约信息可用。
<code>ERROR_BUFFER_OVERFLOW</code>	<code>SizePointer</code> 参数指示的缓冲区大小太小，无法保存适配器信息或 <code>AdapterAddresses</code> 参数为 <code>NULL</code> 。 <code>SizePointer</code> 参数返回的指向所需的缓冲区大小，以保存适配器信息。
<code>ERROR_INVALID_PARAMETER</code>	其中一个参数无效。对于以下任一情况，将返回此错误： <code>SizePointer</code> 参数为 <code>NULL</code> 、 <code>Address</code> 参数不 <code>AF_INET</code> 、 <code>AF_INET6</code> 或 <code>AF_UNSPEC</code> ，或者请求的参数的地址信息大于 <code>ULONG_MAX</code> 。
<code>ERROR_NOT_ENOUGH_MEMORY</code>	内存资源不足，无法完成操作。
<code>ERROR_NO_DATA</code>	找不到所请求参数的地址。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

The

`GetAdaptersAddresses` 函数可以检索 IPv4 和 IPv6 地址的信息。

地址作为 `adapterAddresses` 参数指向的缓冲区中 [IP_ADAPTER_ADDRESSES](#) 结构的链接列表返回。调用 `GetAdaptersAddresses` 函数的应用程序必须分配返回 `AdapterAddresses` 参数指向 [IP_ADAPTER_ADDRESSES](#) 结构所需的内存量。当不再需要这些返回的结构时，应用程序应释放分配的内存。这可以通过调用 [HeapAlloc](#) 函数来分配内存，然后调用 [HeapFree](#) 函数来释放分配的内存，如示例代码所示。只要分配和可用函数使用相同的函数系列，就可以使用其他内存分配和可用函数。

`GetAdaptersAddresses` 仅作为同步函数调用实现。`GetAdaptersAddresses` 函数需要大量的网络资源和时间才能完成，因为必须遍历所有低级别的网络接口表。

可用于确定返回 `AdapterAddresses` 参数指向的 [IP_ADAPTER_ADDRESSES](#) 结构所需的内存的一种方法是，在第一次调用 `GetAdaptersAddresses` 函数时传递 `SizePointer` 参数中指示的缓冲区大小过小，因此函数将失败并 `ERROR_BUFFER_OVERFLOW`。当返回值 `ERROR_BUFFER_OVERFLOW` 时，`SizePointer` 参数返回的指向保存适配器信息所需的缓冲区大小。请注意，如果添加或删除了适配器地址，`AdapterAddresses` 参数指向的

IP_ADAPTER_ADDRESSES 结构所需的缓冲区大小可以在对 **GetAdaptersAddresses** 函数的后续调用之间更改。但是，强烈建议不要使用 **GetAdaptersAddresses** 函数的此方法。此方法需要多次调用 **GetAdaptersAddresses** 函数。

调用 **GetAdaptersAddresses** 函数的建议方法是预先分配 *AdapterAddresses* 参数指向的 15KB 工作缓冲区。在典型计算机上，这大大降低了 **GetAdaptersAddresses** 函数返回 **ERROR_BUFFER_OVERFLOW** 的可能性，这需要多次调用 **GetAdaptersAddresses** 函数。示例代码演示了此使用方法。

在 Windows 10 之前的版本中，可从“网络连接”文件夹控制此函数返回的列表中适配器的显示顺序：从“高级”菜单中选择“高级设置”菜单项。从 Windows 10 开始，适配器在列表中出现的顺序由 IPv4 或 IPv6 路由指标确定。

如果设置了 **GAA_FLAG_INCLUDE_ALL_INTERFACES**，则将检索所有 NDIS 适配器，即使与未绑定到 *Family* 参数中指定的地址系列的适配器关联的地址也是如此。如果未设置此标志，则仅返回绑定到为 *Family* 参数中指定的地址系列启用的适配器的地址。

IP_ADAPTER_ADDRESSES 结构的大小在 Windows XP 上更改了 Service Pack 1 (SP1) 及更高版本。已向此结构添加了几个附加成员。**IP_ADAPTER_ADDRESSES** 结构的大小也在 Windows Vista 及更高版本上进行了更改。向此结构添加了一些附加成员。Windows Vista Service Pack 1 (SP1) 及更高版本以及 Windows Server 2008 及更高版本上，**IP_ADAPTER_ADDRESSES** 结构的大小也发生了变化。向此结构添加了一个附加成员。*AdapterAddresses* 参数指向的缓冲区中结构链接列表中返回的 **IP_ADAPTER_ADDRESSES** 结构的 **Length** 成员应用于确定正在使用哪个版本的 **IP_ADAPTER_ADDRESSES** 结构。

GetIpAddrTable 函数检索本地计算机上的接口到 IPv4 地址映射表，并在 **MIB_IPADDRTABLE** 结构中返回此信息。

在为 Windows Server 2003 及更早版本发布的平台软件开发工具包 (SDK) 中，**GetAdaptersAddresses** 函数的返回值定义为 **DWORD**，而不是 **ULONG**。

SOCKET_ADDRESS 结构用于 *AdapterAddresses* 参数指向的 **IP_ADAPTER_ADDRESSES** 结构中。在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 中，头文件的组织已更改，**SOCKET_ADDRESS** 结构在 *Winsock2.h* 头文件自动包含的 *Ws2def.h* 头文件中定义。在针对 Windows Server 2003 和 Windows XP 发布的平台 SDK 上，**SOCKET_ADDRESS** 结构在 *Winsock2.h* 头文件中声明。若要使用 **IP_ADAPTER_ADDRESSES** 结构，*Winsock2.h* 头文件必须包含在 *Iphlpapi.h* 头文件之前。

示例

此示例检索与系统关联的适配器 的 **IP_ADAPTER_ADDRESSES** 结构，并打印每个适配器接口的一些成员。

C++

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

// Link with Iphlpapi.lib
#pragma comment(lib, "IPHLPAPI.lib")

#define WORKING_BUFFER_SIZE 15000
#define MAXTRIES 3

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int __cdecl main(int argc, char **argv)
{
    /* Declare and initialize variables */

    DWORD dwSize = 0;
    DWORD dwRetVal = 0;

    unsigned int i = 0;

    // Set the flags to pass to GetAdaptersAddresses
    ULONG flags = GAA_FLAG_INCLUDE_PREFIX;

    // default to unspecified address family (both)
    ULONG family = AF_UNSPEC;

    LPVOID lpMsgBuf = NULL;

    PIP_ADAPTER_ADDRESSES pAddresses = NULL;
    ULONG outBufLen = 0;
    ULONG Iterations = 0;

    PIP_ADAPTER_ADDRESSES pCurrAddresses = NULL;
    PIP_ADAPTER_UNICAST_ADDRESS pUnicast = NULL;
    PIP_ADAPTER_ANYCAST_ADDRESS pAnycast = NULL;
    PIP_ADAPTER_MULTICAST_ADDRESS pMulticast = NULL;
    IP_ADAPTER_DNS_SERVER_ADDRESS *pDnServer = NULL;
    IP_ADAPTER_PREFIX *pPrefix = NULL;

    if (argc != 2) {
        printf(" Usage: getadapteraddresses family\n");
        printf("           getadapteraddresses 4 (for IPv4)\n");
        printf("           getadapteraddresses 6 (for IPv6)\n");
        printf("           getadapteraddresses A (for both IPv4 and IPv6)\n");
        exit(1);
    }
```

```

if (atoi(argv[1]) == 4)
    family = AF_INET;
else if (atoi(argv[1]) == 6)
    family = AF_INET6;

printf("Calling GetAdaptersAddresses function with family = ");
if (family == AF_INET)
    printf("AF_INET\n");
if (family == AF_INET6)
    printf("AF_INET6\n");
if (family == AF_UNSPEC)
    printf("AF_UNSPEC\n\n");

// Allocate a 15 KB buffer to start with.
outBufLen = WORKING_BUFFER_SIZE;

do {

    pAddresses = (IP_ADAPTER_ADDRESSES *) MALLOC(outBufLen);
    if (pAddresses == NULL) {
        printf
            ("Memory allocation failed for IP_ADAPTER_ADDRESSES
struct\n");
        exit(1);
    }

    dwRetVal =
        GetAdaptersAddresses(family, flags, NULL, pAddresses,
&outBufLen);

    if (dwRetVal == ERROR_BUFFER_OVERFLOW) {
        FREE(pAddresses);
        pAddresses = NULL;
    } else {
        break;
    }

    Iterations++;

} while ((dwRetVal == ERROR_BUFFER_OVERFLOW) && (Iterations <
MAX_TRIES));

if (dwRetVal == NO_ERROR) {
    // If successful, output some information from the data we received
    pCurrAddresses = pAddresses;
    while (pCurrAddresses) {
        printf("\tLength of the IP_ADAPTER_ADDRESS struct: %ld\n",
               pCurrAddresses->Length);
        printf("\tIfIndex (IPv4 interface): %u\n", pCurrAddresses-
>IfIndex);
        printf("\tAdapter name: %s\n", pCurrAddresses->AdapterName);

        pUnicast = pCurrAddresses->FirstUnicastAddress;
        if (pUnicast != NULL) {

```

```

        for (i = 0; pUnicast != NULL; i++)
            pUnicast = pUnicast->Next;
        printf("\tNumber of Unicast Addresses: %d\n", i);
    } else
        printf("\tNo Unicast Addresses\n");

pAnycast = pCurrAddresses->FirstAnycastAddress;
if (pAnycast) {
    for (i = 0; pAnycast != NULL; i++)
        pAnycast = pAnycast->Next;
    printf("\tNumber of Anycast Addresses: %d\n", i);
} else
    printf("\tNo Anycast Addresses\n");

pMulticast = pCurrAddresses->FirstMulticastAddress;
if (pMulticast) {
    for (i = 0; pMulticast != NULL; i++)
        pMulticast = pMulticast->Next;
    printf("\tNumber of Multicast Addresses: %d\n", i);
} else
    printf("\tNo Multicast Addresses\n");

pDnServer = pCurrAddresses->FirstDnsServerAddress;
if (pDnServer) {
    for (i = 0; pDnServer != NULL; i++)
        pDnServer = pDnServer->Next;
    printf("\tNumber of DNS Server Addresses: %d\n", i);
} else
    printf("\tNo DNS Server Addresses\n");

printf("\tDNS Suffix: %wS\n", pCurrAddresses->DnsSuffix);
printf("\tDescription: %wS\n", pCurrAddresses->Description);
printf("\tFriendly name: %wS\n", pCurrAddresses->FriendlyName);

if (pCurrAddresses->PhysicalAddressLength != 0) {
    printf("\tPhysical address: ");
    for (i = 0; i < (int) pCurrAddresses->PhysicalAddressLength;
         i++) {
        if (i == (pCurrAddresses->PhysicalAddressLength - 1))
            printf("%.2X\n",
                   (int) pCurrAddresses->PhysicalAddress[i]);
        else
            printf("%.2X-", 
                   (int) pCurrAddresses->PhysicalAddress[i]);
    }
}
printf("\tFlags: %ld\n", pCurrAddresses->Flags);
printf("\tMtu: %lu\n", pCurrAddresses->Mtu);
printf("\tIfType: %ld\n", pCurrAddresses->IfType);
printf("\tOperStatus: %ld\n", pCurrAddresses->OperStatus);
printf("\tIpv6IfIndex (IPv6 interface): %u\n",
       pCurrAddresses->Ipv6IfIndex);
printf("\tZoneIndices (hex): ");
for (i = 0; i < 16; i++)
    printf("%lx ", pCurrAddresses->ZoneIndices[i]);

```

```

        printf("\n");

        printf("\tTransmit link speed: %I64u\n", pCurrAddresses-
>TransmitLinkSpeed);
        printf("\tReceive link speed: %I64u\n", pCurrAddresses-
>ReceiveLinkSpeed);

        pPrefix = pCurrAddresses->FirstPrefix;
        if (pPrefix) {
            for (i = 0; pPrefix != NULL; i++)
                pPrefix = pPrefix->Next;
            printf("\tNumber of IP Adapter Prefix entries: %d\n", i);
        } else
            printf("\tNumber of IP Adapter Prefix entries: 0\n");

        printf("\n");

        pCurrAddresses = pCurrAddresses->Next;
    }
} else {
    printf("Call to GetAdaptersAddresses failed with error: %d\n",
           dwRetVal);
    if (dwRetVal == ERROR_NO_DATA)
        printf("\tNo addresses were found for the requested
parameters\n");
    else {

        if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                           FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
                           NULL, dwRetVal, MAKELANGID(LANG_NEUTRAL,
SUBLANG_DEFAULT),
                           // Default language
                           (LPTSTR) & lpMsgBuf, 0, NULL)) {
            printf("\tError: %s", lpMsgBuf);
            LocalFree(lpMsgBuf);
            if (pAddresses)
                FREE(pAddresses);
            exit(1);
        }
    }
}

if (pAddresses) {
    FREE(pAddresses);
}

return 0;
}

```

要求

最低受支持的客户端	Windows XP [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2003 [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIpAddrTable](#)

[HeapAlloc](#)

[HeapFree](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IP_ADAPTER_ADDRESSES](#)

[MIB_IPADDRTABLE](#)

[SOCKET_ADDRESS](#)

[Windows 套接字](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getAdaptersInfo 函数 (iphlpapi.h)

项目2023/08/24

GetAdaptersInfo 函数检索本地计算机的适配器信息。

在 Windows XP 及更高版本上： 使用 [GetAdaptersAddresses](#) 函数而不是 GetAdaptersInfo。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetAdaptersInfo(
    [out]      PIP_ADAPTER_INFO AdapterInfo,
    [in, out]   PULONG          SizePointer
);
```

parameters

[out] AdapterInfo

指向接收IP_ADAPTER_INFO结构链接列表的缓冲区 [的](#) 指针。

[in, out] SizePointer

指向 ULONG 变量的指针，该变量指定 *pAdapterInfo* 参数指向的缓冲区的大小。如果此大小不足以保存适配器信息，则 [GetAdaptersInfo](#) 会用所需的大小填充此变量，并返回 [ERROR_BUFFER_OVERFLOW](#) 错误代码。

返回值

如果函数成功，则返回值 [ERROR_SUCCESS](#) (定义为与 [NO_ERROR](#)) 相同的值。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_BUFFER_OVERFLOW	用于接收适配器信息的缓冲区太小。如果 <i>pOutBufLen</i> 参数指示的缓冲区大小太小而无法保存适配器信息，或者 <i>pAdapterInfo</i> 参数为 NULL 指针，则返回此值。返回此错误代码时， <i>pOutBufLen</i> 参数指向所需的缓冲区大小。

ERROR_INVALID_DATA	检索了无效的适配器信息。
ERROR_INVALID_PARAMETER	其中一个参数无效。如果 <i>pOutBufLen</i> 参数为 NULL 指针，或者调用进程对 <i>pOutBufLen</i> 所指向的内存没有读/写访问权限，或者调用进程对 <i>pAdapterInfo</i> 参数所指向的内存没有写入访问权限，则返回此错误。
ERROR_NO_DATA	本地计算机不存在适配器信息。
ERROR_NOT_SUPPORTED	在本地计算机上运行的操作系统不支持 GetAdaptersInfo 函数。
其他	如果函数失败，请使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetAdaptersInfo](#) 函数只能检索 IPv4 地址的信息。

在 Windows 10 之前的版本中，可从“网络连接”文件夹控制适配器在此函数返回的列表中出现的顺序：从“高级”菜单中选择“高级设置”菜单项。从 Windows 10 开始，顺序未指定。

[GetAdaptersInfo](#) 和 [GetInterfaceInfo](#) 函数不返回有关 IPv4 环回接口的信息。有关环回接口的信息由 [GetIpAddrTable](#) 函数返回。

在 Windows XP 及更高版本上： [GetAdaptersInfo](#) 返回的适配器列表包括单向适配器。若要生成可发送和接收数据的适配器列表，请调用 [GetUniDirectionalAdapterInfo](#)，并从 [GetAdaptersInfo](#) 返回的列表中排除返回的适配器。

示例

此示例检索适配器信息并打印每个适配器的各种属性。

C++

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>
#pragma comment(lib, "IPHLPAPI.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */
```

```
int __cdecl main()
{
    /* Declare and initialize variables */

    // It is possible for an adapter to have multiple
    // IPv4 addresses, gateways, and secondary WINS servers
    // assigned to the adapter.
    //
    // Note that this sample code only prints out the
    // first entry for the IP address/mask, and gateway, and
    // the primary and secondary WINS server for each adapter.

    PIP_ADAPTER_INFO pAdapterInfo;
    PIP_ADAPTER_INFO pAdapter = NULL;
    DWORD dwRetVal = 0;
    UINT i;

    /* variables used to print DHCP time info */
    struct tm newtime;
    char buffer[32];
    errno_t error;

    ULONG ulOutBufLen = sizeof (IP_ADAPTER_INFO);
    pAdapterInfo = (IP_ADAPTER_INFO *) MALLOC(sizeof (IP_ADAPTER_INFO));
    if (pAdapterInfo == NULL) {
        printf("Error allocating memory needed to call GetAdaptersinfo\n");
        return 1;
    }
    // Make an initial call to GetAdaptersInfo to get
    // the necessary size into the ulOutBufLen variable
    if (GetAdaptersInfo(pAdapterInfo, &ulOutBufLen) ==
        ERROR_BUFFER_OVERFLOW) {
        FREE(pAdapterInfo);
        pAdapterInfo = (IP_ADAPTER_INFO *) MALLOC(ulOutBufLen);
        if (pAdapterInfo == NULL) {
            printf("Error allocating memory needed to call
GetAdaptersinfo\n");
            return 1;
        }
    }

    if ((dwRetVal = GetAdaptersInfo(pAdapterInfo, &ulOutBufLen)) ==
        NO_ERROR) {
        pAdapter = pAdapterInfo;
        while (pAdapter) {
            printf("\tComboIndex: \t%d\n", pAdapter->ComboIndex);
            printf("\tAdapter Name: \t%s\n", pAdapter->AdapterName);
            printf("\tAdapter Desc: \t%s\n", pAdapter->Description);
            printf("\tAdapter Addr: \t");
            for (i = 0; i < pAdapter->AddressLength; i++) {
                if (i == (pAdapter->AddressLength - 1))
                    printf("%.2X\n", (int) pAdapter->Address[i]);
                else
                    printf("%.2X-", (int) pAdapter->Address[i]));
            }
        }
    }
}
```

```

    }

    printf("\tIndex: \t%d\n", pAdapter->Index);
    printf("\tType: \t");
    switch (pAdapter->Type) {
        case MIB_IF_TYPE_OTHER:
            printf("Other\n");
            break;
        case MIB_IF_TYPE_ETHERNET:
            printf("Ethernet\n");
            break;
        case MIB_IF_TYPE_TOKENRING:
            printf("Token Ring\n");
            break;
        case MIB_IF_TYPE_FDDI:
            printf("FDDI\n");
            break;
        case MIB_IF_TYPE_PPP:
            printf("PPP\n");
            break;
        case MIB_IF_TYPE_LOOPBACK:
            printf("Lookback\n");
            break;
        case MIB_IF_TYPE_SLIP:
            printf("Slip\n");
            break;
        default:
            printf("Unknown type %ld\n", pAdapter->Type);
            break;
    }

    printf("\tIP Address: \t%s\n",
           pAdapter->IpAddressList.IpAddress.String);
    printf("\tIP Mask: \t%s\n", pAdapter-
>IpAddressList.IpMask.String);

    printf("\tGateway: \t%s\n", pAdapter-
>GatewayList.IpAddress.String);
    printf("\t***\n");

    if (pAdapter->DhcpEnabled) {
        printf("\tDHCP Enabled: Yes\n");
        printf("\t\t DHCP Server: \t%s\n",
               pAdapter->DhcpServer.IpAddress.String);

        printf("\t\t Lease Obtained: ");
        /* Display local time */
        error = _localtime32_s(&newtime, (_time32_t*) &pAdapter-
>LeaseObtained);
        if (error)
            printf("Invalid Argument to _localtime32_s\n");
        else {
            // Convert to an ASCII representation
            error = asctime_s(buffer, 32, &newtime);
            if (error)
                printf("Invalid Argument to asctime_s\n");
        }
    }
}

```

```

        else
            /* asctime_s returns the string terminated by \n\0
 */
            printf("%s", buffer);
    }

    printf("\t Lease Expires: ");
    error = _localtime32_s(&newtime, (_time32_t*) &pAdapter-
>LeaseExpires);
    if (error)
        printf("Invalid Argument to _localtime32_s\n");
    else {
        // Convert to an ASCII representation
        error = asctime_s(buffer, 32, &newtime);
        if (error)
            printf("Invalid Argument to asctime_s\n");
        else
            /* asctime_s returns the string terminated by \n\0
 */
            printf("%s", buffer);
    }
} else
    printf("\tDHCP Enabled: No\n");

if (pAdapter->HaveWins) {
    printf("\tHave Wins: Yes\n");
    printf("\t Primary Wins Server: %s\n",
           pAdapter->PrimaryWinsServer.IpAddress.String);
    printf("\t Secondary Wins Server: %s\n",
           pAdapter->SecondaryWinsServer.IpAddress.String);
} else
    printf("\tHave Wins: No\n");
pAdapter = pAdapter->Next;
printf("\n");
}
} else {
    printf("GetAdaptersInfo failed with error: %d\n", dwRetVal);

}
if (pAdapterInfo)
    FREE(pAdapterInfo);

return 0;
}

```

要求

最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetAdaptersAddresses](#)

[GetInterfaceInfo](#)

[GetIpAddrTable](#)

[GetNumberOfInterfaces](#)

[GetUniDirectionalAdapterInfo](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IP_ADAPTER_INFO](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getAnycastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

GetAnycastIpAddressEntry 函数检索本地计算机上现有 anycast IP 地址条目的信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetAnycastIpAddressEntry(
    [in, out] PMIB_ANYCASTIPADDRESS_ROW Row
);
```

parameters

[in, out] Row

指向 anycast IP 地址条目 的MIB_ANYCASTIPADDRESS_ROW 结构条目的指针。 成功返回后，将使用现有 anycast IP 地址的属性更新此结构。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。 如果由 Row 参数指向的 MIB_ANYCASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	参数不正确。 如果在 Row 参数中传递 NULL 指针，Row 参数指向的MIB_ANYCASTIPADDRESS_ROW的 Address 成员未设置为有效的任意广播 IPv4 或 IPv6 地址，或者未指定 Row 参数指向的MIB_ANYCASTIPADDRESS_ROW的 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误。

ERROR_NOT_FOUND	找不到元素。如果 Row 参数指向的 MIB_ANYCASTIPADDRESS_ROW 结构的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口与 MIB_ANYCASTIPADDRESS_ROW 结构中的 Address 成员中指定的 IP 地址和地址系列不匹配，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW 结构的 Address 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 Address 成员中 指定了 IPv6 地址，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetAnycastIpAddressEntry](#) 函数在 Windows Vista 及更高版本上定义。

[GetAnycastIpAddressEntry](#) 函数用于检索现有的 [MIB_ANYCASTIPADDRESS_ROW](#) 结构条目。

输入时，Row 参数指向的 [MIB_ANYCASTIPADDRESS_ROW](#) 结构中的 Address 成员必须初始化为有效的任意广播 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 Row 参数的 [MIB_ANYCASTIPADDRESS_ROW](#) 结构中的至少一个成员：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则使用此成员来确定接口。如果未为 InterfaceLuid 成员设置任何值，(此成员的值) 设置为零，则接下来使用 InterfaceIndex 成员来确定接口。

在调用成功时输出时，[GetAnycastIpAddressEntry](#) 检索 anycast IP 地址的其他属性，并填写 Row 参数指向的 [MIB_ANYCASTIPADDRESS_ROW](#) 结构。

可以调用 [GetAnycastIpAddressTable](#) 函数来枚举本地计算机上的 anycast IP 地址条目。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows

标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateAnycastIpAddressEntry](#)

[DeleteAnycastIpAddressEntry](#)

[GetAnycastIpAddressTable](#)

[IP 帮助程序函数参考](#)

[MIB_ANYCASTIPADDRESS_ROW](#)

[MIB_ANYCASTIPADDRESS_TABLE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getAnycastIpAddressTable 函数 (netioapi.h)

项目2023/08/25

GetAnycastIpAddressTable 函数检索本地计算机上的 anycast IP 地址表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetAnycastIpAddressTable(
    [in] ADDRESS_FAMILY Family,
    [out] PMIB_ANYCASTIPADDRESS_TABLE *Table
);
```

parameters

[in] Family

要检索的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和PF_ 协议系列常量的值 (相同，例如，AF_INET 和 PF_INET)，因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织已更改，并且 Ws2def.h 头文件中定义了此成员的可能值。请注意，Ws2def.h 头文件自动包含在 Winsock2.h 中，不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	未指定地址系列。指定此参数后，此函数将返回包含 IPv4 和 IPv6 条目的任播 IP 地址表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数后，此函数将返回仅包含 IPv4 条目的任意广播 IP 地址表。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数后，此函数返回仅包含 IPv6 条目的任意广播 IP 地址表。

[out] Table

指向 [MIB_ANYCASTIPADDRESS_TABLE](#) 结构的指针，该结构包含本地计算机上的任意播 IP 地址条目的表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Table</i> 参数中传递 NULL 指针，或者未将 <i>Family</i> 参数指定为 AF_INET、AF_INET6 或 AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	可用内存资源不足，无法完成该操作。
ERROR_NOT_FOUND	未找到 <i>Family</i> 参数中指定的任何播 IP 地址条目。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Family 参数中指定了AF_INET ，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 Family 参数中指定了AF_INET6 ，也会返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`GetAnycastIpAddressTable` 函数在 Windows Vista 及更高版本上定义。

The

`GetAnycastIpAddressTable` 函数枚举本地系统上的任意广播 IP 地址，并在 [MIB_ANYCASTIPADDRESS_TABLE](#) 结构中返回此信息。

anycast IP 地址条目在 *Table* 参数指向的缓冲区中的[MIB_ANYCASTIPADDRESS_TABLE](#) 结构中返回。[MIB_ANYCASTIPADDRESS_TABLE](#) 结构包含一个任意广播 IP 地址条目计数和每个 anycast IP 地址条目的[MIB_ANYCASTIPADDRESS_ROW](#) 结构数组。如果不再需要这些返回的结构，请通过调用 [FreeMibTable](#) 释放内存。

Family 参数必须初始化为AF_INET、AF_INET6或AF_UNSPEC。

请注意，*Table* 参数指向的返回[MIB_ANYCASTIPADDRESS_TABLE](#)结构可能包含 *NumEntries* 成员与 [MIB_ANYCASTIPADDRESS_TABLE 结构](#)的*Table* 成员中第一个 [MIB_ANYCASTIPADDRESS_ROW](#)数组条目之间的对齐填充。
[MIB_ANYCASTIPADDRESS_ROW 数组条目](#)之间也可能存在对齐填充。对 [MIB_ANYCASTIPADDRESS_ROW](#) 数组条目的任何访问都应假定可能存在填充。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FreeMibTable](#)

[MIB_ANYCASTIPADDRESS_ROW](#)

[MIB_ANYCASTIPADDRESS_TABLE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getBestInterface 函数 (iphlpapi.h)

项目2023/08/24

GetBestInterface 函数检索具有指定 IPv4 地址的最佳路由的接口的索引。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetBestInterface(
    [in]  IPAddr dwDestAddr,
    [out] PDWORD pdwBestIfIndex
);
```

parameters

[in] dwDestAddr

要检索具有最佳路由的接口的目标 IPv4 地址，采用 [IPAddr](#) 结构的形式。

[out] pdwBestIfIndex

指向 [DWORD](#) 变量的指针，该变量接收接口的索引，该接口具有 *dwDestAddr* 指定的 IPv4 地址的最佳路由。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_CAN_NOT_COMPLETE	操作无法完成。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>pdwBestIfIndex</i> 参数中传递 NULL 指针，或者 <i>pdwBestIfIndex</i> 指向无法写入的内存，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，则返回此错误。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

[GetBestInterface](#) 函数仅适用于 IPv4 地址。 若要与 IPv6 地址一起使用，必须使用 [GetBestInterfaceEx](#)。

有关 [IPAddr](#) 数据类型的信息，请参阅 [Windows 数据类型](#)。 若要在点点十进制表示法和 [IPAddr](#) 格式之间转换 IP 地址，请使用 [inet_addr](#) 和 [inet_ntoa](#) 函数。

在 Windows Vista 及更高版本上，ip 帮助程序在内部将 *pdwBestIfIndex* 参数视为指向 [NET_IFINDEX](#) 数据类型的指针。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetBestInterfaceEx](#)

[GetBestRoute](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IPAddr](#)

[MIB_BEST_IF](#)

[Windows 数据类型](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

getBestInterfaceEx 函数 (iphlpapi.h)

项目2023/08/24

GetBestInterfaceEx 函数检索具有指定 IPv4 或 IPv6 地址的最佳路由的接口的索引。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetBestInterfaceEx(
    [in]  sockaddr *pDestAddr,
    [out] PDWORD    pdwBestIfIndex
);
```

parameters

[in] pDestAddr

要检索具有最佳路由的接口的目标 IPv6 或 IPv4 地址，采用 `sockaddr` 结构的形式。

[out] pdwBestIfIndex

指向接口索引的指针，该接口具有到 *pDestAddr* 指定的 IPv6 或 IPv4 地址的最佳路由。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_CAN_NOT_COMPLETE	操作无法完成。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>pdwBestIfIndex</i> 参数中传递 NULL 指针，或者 <i>pDestAddr</i> 或 <i>pdwBestIfIndex</i> 参数指向无法访问的内存，则返回此错误。如果 <i>pdwBestIfIndex</i> 参数指向无法写入的内存，也可能会返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 <i>pDestAddr</i> 参数中指定了 IPv4 地址，或者本地计算机上没

有 IPv6 堆栈，并且 *pDestAddr* 参数中指定了 IPv6 地址，则返回此错误。

其他

使用 [FormatMessage](#) 函数获取返回错误的消息字符串。

注解

[GetBestInterfaceEx](#) 函数与 [GetBestInterface](#) 函数的不同之处在于，它可以与 IPv4 或 IPv6 地址一起使用。

pDestAddr 参数指向的 `sockaddr` 结构的 `Family` 成员必须设置为以下值之一： `AF_INET` 或 `AF_INET6`。

在 Windows Vista 及更高版本上，ip 帮助程序在内部将 *pdwBestIfIndex* 参数视为指向 `NET_IFINDEX` 数据类型的指针。

要求

最低受支持的客户端	Windows XP [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2003 [桌面应用 UWP 应用]
目标平台	Windows
标头	<code>iphlpapi.h</code>
Library	<code>iphlpapi.lib</code>
DLL	<code>iphlpapi.dll</code>

另请参阅

[GetBestInterface](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_BEST_IF](#)

[sockaddr](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getBestRoute 函数 (iphlpapi.h)

项目2023/08/24

GetBestRoute 函数检索到指定目标 IP 地址的最佳路由。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetBestRoute(
    [in]    DWORD          dwDestAddr,
    [in]    DWORD          dwSourceAddr,
    [out]   PMIB_IPFORWARDROW pBestRoute
);
```

parameters

[in] dwDestAddr

要为其获取最佳路由的目标 IP 地址。

[in] dwSourceAddr

源 IP 地址。此 IP 地址对应于本地计算机上的接口。如果存在指向目标地址的多个最佳路由，该函数会选择使用此接口的路由。

此参数是可选的。调用方可以指定此参数的零。

[out] pBestRoute

指向 MIB_IPFORWARDROW 结构的指针，该结构包含 dwDestAddr 指定的 IP 地址的最佳路由。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetBestInterface](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPFORWARDROW](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getBestRoute2 函数 (netioapi.h)

项目2023/08/25

GetBestRoute2 函数检索本地计算机上的 IP 路由条目，以获得到指定目标 IP 地址的最佳路由。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetBestRoute2(
    [in, optional] NET_LUID             *InterfaceLuid,
    [in]          NET_IFINDEX         InterfaceIndex,
    [in]          const SOCKADDR_INET *SourceAddress,
    [in]          const SOCKADDR_INET *DestinationAddress,
    [in]          ULONG               AddressSortOptions,
    [out]         PMIB_IPFORWARD_ROW2 BestRoute,
    [out]         SOCKADDR_INET      *BestSourceAddress
);
```

parameters

[in, optional] InterfaceLuid

本地唯一标识符 (LUID) 指定与 IP 路由条目关联的网络接口。

[in] InterfaceIndex

用于指定与 IP 路由条目关联的网络接口的本地索引值。当网络适配器被禁用并随后启用时，或者在其他情况下，此索引值可能会更改，并且不应被视为永久性。

[in] SourceAddress

源 IP 地址。此参数可以省略并作为 NULL 指针传递。

[in] DestinationAddress

目标 IP 地址。

[in] AddressSortOptions

影响 IP 地址排序方式的一组选项。当前未使用此参数。

[out] BestRoute

指向 [MIB_IPFORWARD_ROW2](#) 的指针，用于从源 IP 地址到目标 IP 地址的最佳路由。

[out] BestSourceAddress

指向最佳源 IP 地址的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>DestinationAddress</i> 、 <i>BestSourceAddress</i> 或 <i>BestRoute</i> 参数中传递 NULL 指针，则返回此错误。如果 <i>DestinationAddress</i> 参数未指定 IPv4 或 IPv6 地址和系列，也会返回此错误。
ERROR_FILE_NOT_FOUND	找不到指定的接口。如果找不到 <i>InterfaceLuid</i> 或 <i>InterfaceIndex</i> 参数指定的网络接口，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，且在 <i>DestinationAddress</i> 参数中指定了 IPv4 地址和系列，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并在 <i>DestinationAddress</i> 参数中指定了 IPv6 地址和系列，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

GetBestRoute2 函数在 Windows Vista 及更高版本上定义。

GetBestRoute2 函数用于检索[MIB_IPFORWARD_ROW2](#)结构条目，以便从源 IP 地址到目标 IP 地址的最佳路由。

输入时，*DestinationAddress* 参数必须初始化为有效的 IPv4 或 IPv6 地址和系列。输入时，*SourceAddress* 参数可以初始化为首选 IPv4 或 IPv6 地址和系列。此外，必须至少初始化以下参数之一：*InterfaceLuid* 或 *InterfaceIndex*。

参数按上面列出的顺序使用。因此，如果指定了 *InterfaceLuid*，则使用此成员来确定接口。如果没有为 *InterfaceLuid* 成员设置值，(此成员的值) 设置为零，则接下来使用 *InterfaceIndex* 成员来确定接口。

当调用成功时，`GetBestRoute2` 在输出中检索并 `MIB_IPFORWARD_ROW2` 结构，以便从源 IP 地址到目标 IP 地址的最佳路由。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	<code>netioapi.h</code> (包括 <code>Iphlpapi.h</code>)
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

请参阅

[CreateIpForwardEntry2](#)

[DeleteIpForwardEntry2](#)

[GetIpForwardEntry2](#)

[GetIpForwardTable2](#)

[InitializeIpForwardEntry](#)

[MIB_IPFORWARD_ROW2](#)

[MIB_IPFORWARD_TABLE2](#)

[NotifyRouteChange2](#)

[SetIpForwardEntry2](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

getDefaultValueId 函数 (netioapi.h)

项目2023/08/25

GetDefaultCompartmnetId 函数检索本地计算机的默认网络路由舱标识符。

语法

C++

```
IPHLPAPI_DLL_LINKAGE NET_IF_COMPARTMENT_ID NETIOAPI_API_
GetDefaultCompartmnetId();
```

返回值

如果函数成功，则返回值为默认的隔离舱 ID。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
NET_IF_COMPARTMENT_ID_UNSPECIFIED	函数未能获取隔离舱 ID。

要求

最低受支持的客户端	Windows 10 [仅限桌面应用]
最低受支持的服务器	Windows Server 2016 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

getExtendedTcpTable 函数 (iphlpapi.h)

项目2023/08/24

GetExtendedTcpTable 函数检索包含应用程序可用的 TCP 终结点列表的表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetExtendedTcpTable(
    [out]     PVOID          pTcpTable,
    [in, out]  PDWORD        pdwSize,
    [in]      BOOL           bOrder,
    [in]      ULONG          ulAf,
    [in]      TCP_TABLE_CLASS TableClass,
    [in]      ULONG          Reserved
);
```

parameters

[out] pTcpTable

指向表结构的指针，该表结构包含可用于应用程序的筛选 TCP 终结点。有关如何根据特定输入参数组合确定返回的表类型的信息，请参阅本文档后面的备注部分。

[in, out] pdwSize

pTcpTable 中返回的结构的估计大小（以字节为单位）。如果此值设置得太小，则此函数将返回ERROR_INSUFFICIENT_BUFFER，并且此字段将包含结构的正确大小。

[in] bOrder

一个值，该值指定是否应对 TCP 连接表进行排序。如果此参数设置为 TRUE，则表中的 TCP 终结点按升序排序，从最低本地 IP 地址开始。如果此参数设置为 FALSE，则表中的 TCP 终结点将按检索顺序显示。

在对 TCP 终结点进行排序时，() 列出的 (比较以下值：

1. 本地 IP 地址
2. 当 *ulAf* 参数设置为 AF_INET6 时，本地范围 ID (适用)
3. 本地 TCP 端口
4. 远程 IP 地址
5. 当 *ulAf* 参数设置为 AF_INET6 时，远程范围 ID (适用)

6. 远程 TCP 端口

[in] *ulAf*

TCP 终结点使用的 IP 版本。

值	含义
AF_INET	使用 IPv4。
AF_INET6	使用 IPv6。

[in] *TableClass*

要检索的 TCP 表结构的类型。此参数可以是 [TCP_TABLE_CLASS](#) 枚举中的值之一。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，[TCP_TABLE_CLASS](#) 枚举在 *Iprtrmib.h* 头文件中定义，而不是在 *Iphlpapi.h* 头文件中定义。

[TCP_TABLE_CLASS](#) 枚举值与 *ulAf* 参数的值组合在一起，以确定要检索的扩展 TCP 信息。

[in] *Reserved*

保留。此值必须为零。

返回值

如果调用成功，则返回 **NO_ERROR** 的值。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	为表分配的空间不足。表的大小在 <i>pdwSize</i> 参数中返回，并且必须在后续调用此函数时使用，才能成功检索表。 如果 <i>pTcpTable</i> 参数为 NULL ，也会返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>TableClass</i> 参数包含 TCP_TABLE_CLASS 枚举中未定义的值，则返回此错误。

注解

此函数返回的表类型取决于 *ulAf* 参数和 *TableClass* 参数的特定组合。

当 *uIAf* 参数设置为 AF_INET 时，下表指示在 *pTcpTable* 参数指向的每个可能的 TableClass 值的结构中检索的 TCP 表类型。

TableClass 值	<i>pTcpTable</i> 结构
TCP_TABLE_BASIC_ALL	MIB_TCPTABLE
TCP_TABLE_BASIC_CONNECTIONS	MIB_TCPTABLE
TCP_TABLE_BASIC_LISTENER	MIB_TCPTABLE
TCP_TABLE_OWNER_MODULE_ALL	MIB_TCPTABLE_OWNER_MODULE
TCP_TABLE_OWNER_MODULE_CONNECTIONS	MIB_TCPTABLE_OWNER_MODULE
TCP_TABLE_OWNER_MODULE_LISTENER	MIB_TCPTABLE_OWNER_MODULE
TCP_TABLE_OWNER_PID_ALL	MIB_TCPTABLE_OWNER_PID
TCP_TABLE_OWNER_PID_CONNECTIONS	MIB_TCPTABLE_OWNER_PID
TCP_TABLE_OWNER_PID_LISTENER	MIB_TCPTABLE_OWNER_PID

当 *uIAf* 参数设置为 AF_INET6 时，下表指示在 *pTcpTable* 参数指向的每个可能的 TableClass 值的结构中检索的 TCP 表类型。

TableClass 值	<i>pTcpTable</i> 结构
TCP_TABLE_OWNER_MODULE_ALL	MIB_TCP6TABLE_OWNER_MODULE
TCP_TABLE_OWNER_MODULE_CONNECTIONS	MIB_TCP6TABLE_OWNER_MODULE
TCP_TABLE_OWNER_MODULE_LISTENER	MIB_TCP6TABLE_OWNER_MODULE
TCP_TABLE_OWNER_PID_ALL	MIB_TCP6TABLE_OWNER_PID
TCP_TABLE_OWNER_PID_CONNECTIONS	MIB_TCP6TABLE_OWNER_PID
TCP_TABLE_OWNER_PID_LISTENER	MIB_TCP6TABLE_OWNER_PID

不支持将 *uIAf* 参数设置为 AF_INET6 且 TableClass 设置为 TCP_TABLE_BASIC_LISTENER、TCP_TABLE_BASIC_CONNECTIONS 或 TCP_TABLE_BASIC_ALL 调用的 GetExtendedTcpTable 函数，并返回 ERROR_NOT_SUPPORTED。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改。各种 MIB_TCPTABLE 结构在 *Tcpmib.h* 头文件中定义，而不是在 *lprtrmib.h* 头文件中定

义。请注意，*Tcpmib.h* 头文件会自动包含在 *Iphlpapi.h* 头文件中，*Iprtrmib.h* 中会自动包含该文件。不应直接使用 *Tcpmib.h* 和 *Iprtrmib.h* 头文件。

要求

最低受支持的客户端	Windows Vista、Windows XP 和 SP2 [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[MIB_TCP6TABLE](#)

[MIB_TCP6TABLE_OWNER_MODULE](#)

[MIB_TCP6TABLE_OWNER_PID](#)

[MIB_TCPTABLE](#)

[MIB_TCPTABLE_OWNER_MODULE](#)

[MIB_TCPTABLE_OWNER_PID](#)

[TCP_TABLE_CLASS](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

getExtendedUdpTable 函数 (iphlpapi.h)

项目2023/08/24

GetExtendedUdpTable 函数检索包含应用程序可用的 UDP 终结点列表的表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetExtendedUdpTable(
    [out]     PVOID          pUdpTable,
    [in, out] PDWORD         pdwSize,
    [in]      BOOL           bOrder,
    [in]      ULONG          ulAf,
    [in]      UDP_TABLE_CLASS TableClass,
    [in]      ULONG          Reserved
);
```

parameters

[out] pUdpTable

指向表结构的指针，该表结构包含可用于应用程序的筛选 UDP 终结点。有关如何根据特定输入参数组合确定返回的表类型的信息，请参阅本文档后面的备注部分。

[in, out] pdwSize

pUdpTable 中返回的结构的估计大小（以字节为单位）。如果此值设置得太小，则此函数将返回ERROR_INSUFFICIENT_BUFFER，并且此字段将包含结构的正确大小。

[in] bOrder

一个值，该值指定是否应对 UDP 终结点表进行排序。如果此参数设置为 TRUE，则表中的 UDP 终结点按升序排序，从最低本地 IP 地址开始。如果此参数设置为 FALSE，则表中的 UDP 终结点按检索顺序显示。

对 UDP 终结点进行排序时，将比较下列值：

1. 本地 IP 地址
2. 当 *ulAf* 参数设置为 AF_INET6 时，本地范围 ID (适用)
3. 本地 UDP 端口

[in] ulAf

UDP 终结点使用的 IP 版本。

值	含义
AF_INET	使用 IPv4。
AF_INET6	使用 IPv6。

[in] *TableClass*

要检索的 UDP 表结构的类型。 此参数可以是 [UDP_TABLE_CLASS](#) 枚举中的值之一。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织已更改，[UDP_TABLE_CLASS](#) 枚举在 *Iprtrmib.h* 头文件中定义，而不是在 *Iphlpapi.h* 头文件中定义。

[UDP_TABLE_CLASS](#) 枚举值与 *ulAf* 参数的值相结合，以确定要检索的扩展 UDP 信息。

[in] *Reserved*

保留。 此值必须为零。

返回值

如果调用成功，则返回 [NO_ERROR](#) 的值。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	为表分配的空间不足。 表的大小在 <i>pdwSize</i> 参数中返回，并且必须在后续调用此函数时使用，才能成功检索表。 如果 <i>pUdpTable</i> 参数为 NULL ，也会返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果 <i>TableClass</i> 参数包含 UDP_TABLE_CLASS 枚举中未定义的值，则返回此错误。

注解

此函数返回的表类型取决于 *ulAf* 参数和 *TableClass* 参数的特定组合。

当 *ulAf* 参数设置为 **AF_INET** 时，下表指示在 *pUdpTable* 参数指向的每个可能的 *TableClass* 值的结构中检索的 UDP 表类型。

TableClass 值

pUdpTable 结构

UDP_TABLE_BASIC	MIB_UDPTABLE
UDP_TABLE_OWNER_MODULE	MIB_UDPTABLE_OWNER_MODULE
UDP_TABLE_OWNER_PID	MIB_UDPTABLE_OWNER_PID

当 *ulAf* 参数设置为 AF_INET6 时，下表指示在 *pUdpTable* 参数指向的每个可能的 *TableClass* 值的结构中检索的 TCP 表类型。

<i>TableClass</i> 值	<i>pUdpTable</i> 结构
UDP_TABLE_BASIC	MIB_UDP6TABLE
UDP_TABLE_OWNER_MODULE	MIB_UDP6TABLE_OWNER_MODULE
UDP_TABLE_OWNER_PID	MIB_UDP6TABLE_OWNER_PID

调用 [时 getExtendedUdpTable](#) 函数，将 *ulAf* 参数设置为 AF_INET6 并且 *TableClass* 设置为 UDP_TABLE_BASIC 仅在 Windows Vista 及更高版本上受支持。

在 Service Pack 1 (SP1) 的 Windows Server 2003 和 Service Pack 2 (SP2) 的 Windows XP 上，调用的 [GetExtendedUdpTable](#) 函数 将 *ulAf* 参数设置为 AF_INET6， *TableClass* 设置为 UDP_TABLE_BASIC 失败并返回 ERROR_NOT_SUPPORTED。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改。各种 [MIB_UDPTABLE](#) 结构在 *Udpmib.h* 头文件中定义，而不是 在 *Iprtrmib.h* 头文件中定义。请注意，*Udpmib.h* 头文件会自动包含在 *Iphlpapi.h* 头文件中。*Iprtrmib.h* 中会自动包含该头文件。不应直接使用 *Udpmib.h* 和 *Iprtrmib.h* 头文件。

要求

最低受支持的客户端	Windows Vista、Windows XP 和 SP2 [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [桌面应用 UWP 应用]
目标平台	Windows
标头	<i>iphlpapi.h</i>
Library	<i>iphlpapi.lib</i>
DLL	<i>iphlpapi.dll</i>

另请参阅

[MIB_UDP6TABLE](#)

[MIB_UDP6TABLE_OWNER_MODULE](#)

[MIB_UDP6TABLE_OWNER_PID](#)

[MIB_UDPTABLE](#)

[MIB_UDPTABLE_OWNER_MODULE](#)

[MIB_UDPTABLE_OWNER_PID](#)

[UDP_TABLE_CLASS](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

GetFriendlyIfIndex 函数 (iphlpapi.h)

项目2023/08/24

GetFriendlyIfIndex 函数采用接口索引并返回向后兼容的接口索引，即仅使用较低 24 位的索引。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetFriendlyIfIndex(  
    [in] DWORD IfIndex  
>;
```

parameters

[in] IfIndex

从中派生向后兼容或“友好”接口索引的接口索引。

返回值

仅使用较低 24 位的向后兼容接口索引。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIfEntry](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IFROW](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getIcmpStatistics 函数 (iphlpapi.h)

项目2023/08/24

`GetIcmpStatistics` 函数检索 Internet 控制消息协议 (ICMP) , 以获取本地计算机的 IPv4 统计信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetIcmpStatistics(
    [out] PMIB_ICMP Statistics
);
```

parameters

[out] `Statistics`

指向接收本地计算机的 ICMP 统计信息 的 `MIB_ICMP` 结构的指针。

返回值

如果函数成功，则返回值`NO_ERROR`。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
<code>ERROR_INVALID_PARAMETER</code>	<code>pStats</code> 参数为 <code>NULL</code> , 或者 <code>GetIcmpStatistics</code> 无法写入 <code>pStats</code> 参数指向的内存。
其他	使用 <code>FormatMessage</code> 函数获取返回错误的消息字符串。

注解

`GetIcmpStatistics` 函数返回本地计算机上 IPv4 的 ICMP 统计信息。在 Windows XP 及更高版本上，`GetIpStatisticsEx` 可用于获取本地计算机上 IPv4 或 IPv6 的 ICMP 统计信息。

示例

以下示例检索本地计算机的 IPv4 ICMP 统计信息，并从返回的数据中输出一些信息。

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int main()
{
    DWORD dwRetVal = 0;
    PMIB_ICMP pIcmpStats;

    pIcmpStats = (MIB_ICMP *) MALLOC(sizeof(MIB_ICMP));
    if (pIcmpStats == NULL) {
        wprintf(L"Error allocating memory\n");
        return 1;
    }

    dwRetVal = GetIcmpStatistics(pIcmpStats);
    if (dwRetVal == NO_ERROR) {
        wprintf(L"Number of incoming ICMP messages: %ld\n",
               pIcmpStats->stats.icmpInStats.dwMsgs);
        wprintf(L"Number of incoming ICMP errors received: %ld\n",
               pIcmpStats->stats.icmpInStats.dwErrors);
        wprintf(L"Number of outgoing ICMP messages: %ld\n",
               pIcmpStats->stats.icmpOutStats.dwMsgs);
        wprintf(L"Number of outgoing ICMP errors sent: %ld\n",
               pIcmpStats->stats.icmpOutStats.dwErrors);
    } else {
        wprintf(L"GetIcmpStatistics failed with error: %ld\n", dwRetVal);
    }

    if (pIcmpStats)
        FREE(pIcmpStats);

    return 0;
}
```

要求

最低受支持的客户端	Windows 2000 专业版 [桌面应用 UWP 应用]
最低受支持的服务器	Windows 2000 Server [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIpStatistics](#)

[GetIpStatisticsEx](#)

[GetTcpStatistics](#)

[GetUdpStatistics](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_ICMP](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

GetIcmpStatisticsEx 函数 (iphlpapi.h)

项目2023/08/24

GetIcmpStatisticsEx 函数检索 Internet 控制消息协议 (ICMP) 本地计算机的统计信息。GetIcmpStatisticsEx 函数能够检索 IPv6 ICMP 统计信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetIcmpStatisticsEx(
    [out] PMIB_ICMP_EX Statistics,
    [in]   ULONG       Family
);
```

parameters

[out] Statistics

指向包含本地计算机的 ICMP 统计信息 的MIB_ICMP_EX 结构的指针。

[in] Family

要检索其 ICMP 统计信息的协议系列。 必须是下列选项之一：

值	含义
AF_INET	Internet 协议版本 4 (IPv4)。
AF_INET6	Internet 协议版本 6 (IPv6)。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	pStats 参数为 NULL 或未指向有效内存，或者 dwFamily 参数不是有效值。

ERROR_NOT_SUPPORTED	执行函数调用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetIpStatisticsEx](#) 可用于获取本地计算机上的 IPv4 或 IPv6 的 ICMP 统计信息。

[GetIcmpStatistics](#) 函数仅返回本地计算机上的 IPv4 的 ICMP 统计信息。

要求

最低受支持的客户端	Windows XP [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2003 [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[GetIcmpStatistics](#)

[GetTcpStatisticsEx](#)

[GetUdpStatisticsEx](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_ICMP_EX](#)

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获得帮助

getIfEntry 函数 (iphlpapi.h)

项目2023/08/24

GetIfEntry 函数检索本地计算机上指定接口的信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetIfEntry(
    [in, out] PMIB_IFROW pIfRow
);
```

parameters

[in, out] pIfRow

指向 MIB_IFROW 结构的指针，该结构在成功返回时接收本地计算机上的接口的信息。 输入时，将 MIB_IFROW 的 dwIndex 成员设置为要检索其信息的接口的索引。 dwIndex 的值必须由以前调用 GetIfTable、GetIfTable2 或 GetIfTable2Ex 函数检索。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_CAN_NOT_COMPLETE	无法完成请求。这是内部错误。
ERROR_INVALID_DATA	数据无效。如果 pIfRow 参数指向的 MIB_IFROW 结构的 dwIndex 成员指定的网络接口索引不是本地计算机上的有效接口索引，则返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 pIfRow 参数中传递 NULL 指针，则返回此错误。
ERROR_NOT_FOUND	找不到指定的接口。如果找不到 pIfRow 参数指向的 MIB_IFROW 结构的 dwIndex 成员指定的网络接口索引，则返回此错误。

ERROR_NOT_SUPPORTED	不支持该请求。如果未在本地计算机上配置 IPv4，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`GetIfEntry` 函数检索本地计算机上的接口的信息。

`pIfRow` 参数指向的 [MIB_IFROW](#) 结构中的 `dwIndex` 成员必须初始化为由先前调用 `GetIfTable`、`GetIfTable2` 或 `GetIfTable2Ex` 函数检索的有效网络接口索引。

如果 `pIfRow` 参数指向的 [MIB_IFROW](#) 的 `dwIndex` 成员与本地计算机上的现有接口索引不匹配，则 `GetIfEntry` 函数将失败。

示例

以下示例从接口表中检索条目，并输出可用于该条目的一些信息。

C++

```
#include <winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "IPHLPAPI.lib")

#include <iphlpapi.h>

#include <stdio.h>
#include <stdlib.h>

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int main()
{
    // Declare and initialize variables.

    // Declare and initialize variables.
    DWORD dwSize = 0;
    DWORD dwRetVal = 0;

    unsigned int i, j;

    /* variables used for GetIfTable and GetIfEntry */
    MIB_IFTABLE *pIfTable;
    MIB_IFROW *pIfRow;
```

```

// Allocate memory for our pointers.
pIfTable = (MIB_IFTABLE *) MALLOC(sizeof (MIB_IFTABLE));
if (pIfTable == NULL) {
    printf("Error allocating memory needed to call GetIfTable\n");
    exit (1);
}
// Before calling GetIfEntry, we call GetIfTable to make
// sure there are entries to get and retrieve the interface index.

// Make an initial call to GetIfTable to get the
// necessary size into dwSize
dwSize = sizeof (MIB_IFTABLE);
if (GetIfTable(pIfTable, &dwSize, 0) == ERROR_INSUFFICIENT_BUFFER) {
    FREE(pIfTable);
    pIfTable = (MIB_IFTABLE *) MALLOC(dwSize);
    if (pIfTable == NULL) {
        printf("Error allocating memory\n");
        exit (1);
    }
}
// Make a second call to GetIfTable to get the actual
// data we want.
if ((dwRetVal = GetIfTable(pIfTable, &dwSize, 0)) == NO_ERROR) {
    if (pIfTable->dwNumEntries > 0) {
        pIfRow = (MIB_IFROW *) MALLOC(sizeof (MIB_IFROW));
        if (pIfRow == NULL) {
            printf("Error allocating memory\n");
            if (pIfTable != NULL) {
                FREE(pIfTable);
                pIfTable = NULL;
            }
            exit (1);
        }

        printf("\tNum Entries: %ld\n\n", pIfTable->dwNumEntries);
        for (i = 0; i < pIfTable->dwNumEntries; i++) {
            pIfRow->dwIndex = pIfTable->table[i].dwIndex;
            if ((dwRetVal = GetIfEntry(pIfRow)) == NO_ERROR) {
                printf("\tIndex:\t %d\n", pIfRow->dwIndex);
                printf("\tInterfaceName[%d]:\t ", i);
                if (pIfRow->wszName != NULL)
                    printf("%ws", pIfRow->wszName);
                printf("\n");

                printf("\tDescription[%d]:\t ", i);
                for (j = 0; j < pIfRow->dwDescrLen; j++)
                    printf("%c", pIfRow->bDescr[j]);
                printf("\n");

                printf("\tIndex[%d]:\t\t %d\n", i, pIfRow->dwIndex);

                printf("\tType[%d]:\t\t ", i);
                switch (pIfRow->dwType) {
                    case IF_TYPE_OTHER:

```

```

        printf("Other\n");
        break;
    case IF_TYPE_ETHERNET_CSMACD:
        printf("Ethernet\n");
        break;
    case IF_TYPE_ISO88025_TOKENRING:
        printf("Token Ring\n");
        break;
    case IF_TYPE_PPP:
        printf("PPP\n");
        break;
    case IF_TYPE_SOFTWARE_LOOPBACK:
        printf("Software Lookback\n");
        break;
    case IF_TYPE_ATM:
        printf("ATM\n");
        break;
    case IF_TYPE_IEEE80211:
        printf("IEEE 802.11 Wireless\n");
        break;
    case IF_TYPE_TUNNEL:
        printf("Tunnel type encapsulation\n");
        break;
    case IF_TYPE_IEEE1394:
        printf("IEEE 1394 Firewire\n");
        break;
    default:
        printf("Unknown type %ld\n", pIfRow->dwType);
        break;
}

printf("\tMtu[%d]:\t\t %ld\n", i, pIfRow->dwMtu);

printf("\tSpeed[%d]:\t\t %ld\n", i, pIfRow->dwSpeed);

printf("\tPhysical Addr:\t\t ");
if (pIfRow->dwPhysAddrLen == 0)
    printf("\n");
// for (j = 0; j < (int) pIfRow->dwPhysAddrLen; j++) {
for (j = 0; j < pIfRow->dwPhysAddrLen; j++) {
    if (j == (pIfRow->dwPhysAddrLen - 1))
        printf("%.2X\n", (int) pIfRow->bPhysAddr[j]);
    else
        printf("%.2X-", (int) pIfRow->bPhysAddr[j]);
}
printf("\tAdmin Status[%d]:\t %ld\n", i,
       pIfRow->dwAdminStatus);

printf("\tOper Status[%d]:\t ", i);
switch (pIfRow->dwOperStatus) {
case IF_OPER_STATUS_NON_OPERATIONAL:
    printf("Non Operational\n");
    break;
case IF_OPER_STATUS_UNREACHABLE:
    printf("Unreachable\n");
}

```

```

        break;
    case IF_OPER_STATUS_DISCONNECTED:
        printf("Disconnected\n");
        break;
    case IF_OPER_STATUS_CONNECTING:
        printf("Connecting\n");
        break;
    case IF_OPER_STATUS_CONNECTED:
        printf("Connected\n");
        break;
    case IF_OPER_STATUS_OPERATIONAL:
        printf("Operational\n");
        break;
    default:
        printf("Unknown status %ld\n",
               pIfRow->dwOperStatus);
        break;
    }
    printf("\n");
}

else {
    printf("GetIfEntry failed for index %d with error:
%ld\n",
           i, dwRetVal);
    // Here you can use FormatMessage to find out why
    // it failed.

    }
}
} else {
    printf("\tGetIfTable failed with error: %ld\n", dwRetVal);
}

}

exit (0);
}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h

Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIfEntry2](#)

[GetIfTable](#)

[GetIfTable2](#)

[GetIfTable2Ex](#)

[GetNumberOfInterfaces](#)

[IP 帮助程序函数参考](#)

[MIB_IFROW](#)

[MIB_IFTABLE](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

[SetIfEntry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIfEntry2 函数 (netioapi.h)

项目2023/08/25

GetIfEntry2 函数检索本地计算机上指定接口的信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIfEntry2(
    PMIB_IF_ROW2 Row
);
```

parameters

Row

指向 [MIB_IF_ROW2](#) 结构的指针，该结构在成功返回时接收本地计算机上的接口的信息。输入时，必须将 [MIB_IF_ROW2](#) 的 `InterfaceLuid` 或 `InterfaceIndex` 成员设置为要为其检索信息的接口。

返回值

如果函数成功，则返回值 `NO_ERROR`。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
<code>ERROR_FILE_NOT_FOUND</code>	系统找不到指定的文件。如果 <code>Row</code> 参数指向的 MIB_IF_ROW2 的 <code>InterfaceLuid</code> 或 <code>InterfaceIndex</code> 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
<code>ERROR_INVALID_PARAMETER</code>	向该函数传递了无效参数。如果在 <code>Row</code> 参数中传递 <code>NULL</code> 参数，则返回此错误。如果未指定 <code>Row</code> 参数指向的 MIB_IF_ROW2 <code>InterfaceLuid</code> 和 <code>InterfaceIndex</code> 成员，也会返回此错误。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

GetIfEntry2 函数在 Windows Vista 及更高版本上定义。

输入时，必须初始化 *Row* 参数中传递的 MIB_IF_ROW2 结构中的至少一个成员：**InterfaceLuid** 或 **InterfaceIndex**。

字段按上面列出的顺序使用。因此，如果指定了 **InterfaceLuid**，则使用此成员来确定接口。如果未为 **InterfaceLuid** 成员设置值（此成员的值设置为零），则接下来使用 **InterfaceIndex** 成员来确定接口。

在输出中，将填充 *Row* 参数指向的 MIB_IF_ROW2 结构的其余字段。

请注意，*Netioapi.h* 头文件自动包含在 *Iphlpapi.h* 头文件中，不应直接使用。

示例

以下示例检索命令行中指定的接口条目，并从检索到的 MIB_IF_ROW2 结构中输出一些值。

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <Windows.h>

#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>

#include <objbase.h>
#include <wtypes.h>
#include <stdio.h>
#include <stdlib.h>

// Need to link with Iphlpapi.lib
#pragma comment(lib, "iphlpapi.lib")

// Need to link with Ole32.lib to print GUID
#pragma comment(lib, "ole32.lib")

void PrintIfEntry2(PMIB_IF_ROW2 pifRow);

int __cdecl wmain(int argc, WCHAR ** argv)
```

```

{

    // Declare and initialize variables

    ULONG retVal = 0;
    ULONG ifIndex;

    MIB_IF_ROW2 ifRow;

    // Make sure the ifRow is zeroed out
    SecureZeroMemory((PVOID) &ifRow, sizeof(MIB_IF_ROW2) );

    // Zero out the MIB_IF_ROW2 struct

    // Validate the parameters
    if (argc < 2) {
        wprintf(L"usage: %s <InterfaceIndex>\n", argv[0]);
        wprintf(L"    Gets the Interface Entry for an interface Index,\n");
        wprintf(L"Example to get the interface at interface index=6\n");
        wprintf(L"        %s 6\n", argv[0]);
        exit(1);
    }

    ifIndex = _wtoi(argv[1]);

    ifRow.InterfaceIndex = ifIndex;

    retVal = GetIfEntry2(&ifRow);

    if (retVal != NO_ERROR) {
        wprintf(L"GetIfEntry returned error: %lu\n", retVal);
        exit(1);
    }
    else
        wprintf(L"GetIfEntry2 function returned okay\n");

    PrintIfEntry2(&ifRow);

    exit(0);
}

// Print some parameters from the MIB_IF_ROW2 structure
void PrintIfEntry2(PMIB_IF_ROW2 pIfRow)
{
    int iRet = 0;
    WCHAR GuidString[40] = { 0 };

    unsigned int j;

    wprintf(L"\tInterfaceIndex:\t %lu\n", pIfRow->InterfaceIndex);

    iRet = StringFromGUID2(pIfRow->InterfaceGuid, (LPOLESTR) & GuidString,
39);
    // For c rather than C++ source code, the above line needs to be
}

```

```
// iRet = StringFromGUID2(&pIfRow->InterfaceGuid, (LPOLESTR)
&GuidString, 39);
if (iRet == 0)
    wprintf(L"StringFromGUID2 failed\n");
else {
    wprintf(L"\tInterfaceGUID: %ws\n", GuidString);
}

wprintf(L"\tAlias:\t %ws", pIfRow->Alias);
wprintf(L"\n");
wprintf(L"\tDescription:\t %ws", pIfRow->Description);
wprintf(L"\n");
wprintf(L"\tPhysical Address:\t ");
if (pIfRow->PhysicalAddressLength == 0)
    wprintf(L"\n");
for (j = 0; j < (int) pIfRow->PhysicalAddressLength; j++) {
    if (j == (pIfRow->PhysicalAddressLength - 1))
        wprintf(L".%2X\n", (int) pIfRow->PhysicalAddress[j]);
    else
        wprintf(L"%2X-", (int) pIfRow->PhysicalAddress[j]);
}
wprintf(L"\tPermanent Physical Address: ");
if (pIfRow->PhysicalAddressLength == 0)
    wprintf(L"\n");
for (j = 0; j < (int) pIfRow->PhysicalAddressLength; j++) {
    if (j == (pIfRow->PhysicalAddressLength - 1))
        wprintf(L".%2X\n", (int) pIfRow->PermanentPhysicalAddress[j]);
    else
        wprintf(L"%2X-", (int) pIfRow->PermanentPhysicalAddress[j]);
}
wprintf(L"\tMtu:\t %lu\n", pIfRow->Mtu);

wprintf(L"\tType:\t ");
switch (pIfRow->Type) {
case IF_TYPE_OTHER:
    wprintf(L"Other\n");
    break;
case IF_TYPE_ETHERNET_CSMACD:
    wprintf(L"Ethernet\n");
    break;
case IF_TYPE_IS088025_TOKENRING:
    wprintf(L"Token Ring\n");
    break;
case IF_TYPE_PPP:
    wprintf(L"PPP\n");
    break;
case IF_TYPE_SOFTWARE_LOOPBACK:
    wprintf(L"Software Lookback\n");
    break;
case IF_TYPE_ATM:
    wprintf(L"ATM\n");
    break;
case IF_TYPE_IEEE80211:
    wprintf(L"IEEE 802.11 Wireless\n");
    break;
```

```

    case IF_TYPE_TUNNEL:
        wprintf(L"Tunnel type encapsulation\n");
        break;
    case IF_TYPE_IEEE1394:
        wprintf(L"IEEE 1394 Firewire\n");
        break;
    default:
        wprintf(L"Unknown type %ld\n", pIfRow->Type);
        break;
}

wprintf(L"\tTunnel Type:\t ");
switch (pIfRow->TunnelType) {
case TUNNEL_TYPE_NONE:
    wprintf(L"Not a tunnel\n");
    break;
case TUNNEL_TYPE_OTHER:
    wprintf(L"None of the known tunnel types\n");
    break;
case TUNNEL_TYPE_DIRECT:
    wprintf(L"Encapsulated directly within IPv4\n");
    break;
case TUNNEL_TYPE_6TO4:
    wprintf
        (L"IPv6 packet encapsulated within IPv4 using 6to4 protocol\n");
    break;
case TUNNEL_TYPE_ISATAP:
    wprintf
        (L"IPv6 packet encapsulated within IPv4 using ISATAP
protocol\n");
    break;
case TUNNEL_TYPE_TEREDO:
    wprintf(L"Teredo encapsulation\n");
    break;
default:
    wprintf(L"Unknown tunnel type %ld\n", pIfRow->TunnelType);
    break;
}

wprintf(L"\tNDIS Media Type:\t ");
switch (pIfRow->MediaType) {
case NdisMedium802_3:
    wprintf(L"Ethernet (802.3)\n");
    break;
case NdisMedium802_5:
    wprintf(L"Token Ring (802.5)\n");
    break;
case NdisMediumFddi:
    wprintf(L"Fiber Distributed Data Interface (FDDI)\n");
    break;
case NdisMediumWan:
    wprintf(L"Wide area network (WAN)\n");
    break;
case NdisMediumLocalTalk:
    wprintf(L"LocalTalk\n");
}

```

```

        break;
    case NdisMediumDix:
        wprintf(L"Ethernet using DIX header format\n");
        break;
    case NdisMediumArcnetRaw:
        wprintf(L"ARCNET\n");
        break;
    case NdisMediumArcnet878_2:
        wprintf(L"ARCNET (878.2)\n");
        break;
    case NdisMediumAtm:
        wprintf(L"ATM\n");
        break;
    case NdisMediumWirelessWan:
        wprintf(L"Wireless WAN\n");
        break;
    case NdisMediumIrda:
        wprintf(L"Infrared (IrDA)\n");
        break;
    case NdisMediumBpc:
        wprintf(L"Broadcast PC\n");
        break;
    case NdisMediumCoWan:
        wprintf(L"Connection-oriented Wide Area Network (CoWAN)\n");
        break;
    case NdisMedium1394:
        wprintf(L"IEEE 1394 (fire wire)\n");
        break;
    case NdisMediumInfiniBand:
        wprintf(L"InfiniBand\n");
        break;
    case NdisMediumTunnel:
        wprintf(L"A Tunnel\n");
        break;
    case NdisMediumNative802_11:
        wprintf(L"Native IEEE 802.11\n");
        break;
    case NdisMediumLoopback:
        wprintf(L"NDIS loopback \n");
        break;
    default:
        wprintf(L"Unknown media type %ld\n", pIfRow->MediaType);
        break;
    }

    printf("\tAdministrative Status:\t ");
    switch (pIfRow->AdminStatus) {
    case NET_IF_ADMIN_STATUS_UP:
        wprintf(L"Interface up and enabled\n");
        break;
    case NET_IF_ADMIN_STATUS_DOWN:
        wprintf(L"Interface down\n");
        break;
    case NET_IF_ADMIN_STATUS_TESTING:
        wprintf(L"Interface in test mode\n");

```

```

        break;
    default:
        wprintf(L"Unknown status %ld\n", pIfRow->AdminStatus);
        break;
    }

    printf("\tMedia connection state:\t");
    switch (pIfRow->MediaConnectState) {
    case MediaConnectStateUnknown:
        wprintf(L"Interface state is unknown\n");
        break;
    case MediaConnectStateConnected:
        wprintf(L"Connected\n");
        break;
    case MediaConnectStateDisconnected:
        wprintf(L"Disconnected\n");
        break;
    default:
        wprintf(L"Unknown state %ld\n", pIfRow->MediaConnectState);
        break;
    }

    wprintf(L"\tTransmit link speed:\t %I64u\n", pIfRow->TransmitLinkSpeed);
    wprintf(L"\tReceive link speed:\t %I64u\n", pIfRow->ReceiveLinkSpeed);

}

```

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[GetIfEntry](#)

[GetIfTable](#)

[GetIfTable2](#)

[GetIfTable2Ex](#)

[IP 帮助程序函数参考](#)

[MIB_IFROW](#)

[MIB_IFTABLE](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

反馈

此页面是否有帮助?

[!\[\]\(e5e8640123db01125102cf61a623ddfa_img.jpg\) 是](#)

[!\[\]\(22bb7f6aa48f53838a8a63e9bdf46190_img.jpg\) 否](#)

[在 Microsoft Q&A 获得帮助](#)

getIfEntry2Ex 函数 (netioapi.h)

项目2023/08/25

GetIfEntry2Ex 函数检索本地计算机上指定接口的指定信息级别。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIfEntry2Ex(
    [in]      MIB_IF_ENTRY_LEVEL Level,
    [in, out] PMIB_IF_ROW2        Row
);
```

parameters

[in] Level

要检索的接口信息的级别。此参数可以是 *Netioapi.h* 头文件中定义的 **MIB_IF_ENTRY_LEVEL** 枚举类型的值之一。

值	含义
MibIfEntryNormal 0	<i>Row</i> 参数指向的 MIB_IF_ROW2 结构成员中返回的统计信息值和状态值从筛选器堆栈顶部返回。
MibIfEntryNormalWithoutStatistics 2	在 <i>Row</i> 参数指向的 MIB_IF_ROW2 结构成员中返回的状态 (值,) 从筛选器堆栈顶部返回。

[in, out] Row

指向 **MIB_IF_ROW2** 结构的指针，成功返回后，该结构接收本地计算机上的接口信息。输入时，必须将**MIB_IF_ROW2**的 **InterfaceLuid** 或 **InterfaceIndex** 成员设置为要检索其信息的接口。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码

说明

ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果由 <i>Row</i> 参数指向的 MIB_IF_ROW2 的 <i>InterfaceLuid</i> 或 <i>InterfaceIndex</i> 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Row</i> 参数中传递 NULL 参数，则返回此错误。如果未指定 <i>Row</i> 参数指向的 MIB_IF_ROW2 的 <i>InterfaceLuid</i> 和 <i>InterfaceIndex</i> 成员，也会返回此错误。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

The

`GetIfEntry2Ex` 函数检索本地系统上指定接口的信息，并在指向的指针中返回此信息 [MIB_IF_ROW2](#) 结构。`GetIfEntry2Ex` 是 `GetIfEntry2` 函数的增强版本，允许选择要检索的接口信息级别。

输入时，必须初始化在 *Row* 参数中传递的 [MIB_IF_ROW2](#) 结构中至少一个以下成员：[InterfaceLuid](#) 或 [InterfaceIndex](#)。

字段按上面列出的顺序使用。因此，如果指定了 [InterfaceLuid](#)，则使用此成员来确定接口。如果没有为 [InterfaceLuid](#) 成员设置值，(此成员的值) 设置为零，则接下来使用 [InterfaceIndex](#) 成员来确定接口。

在输出时，将填充 *Row* 参数指向的 [MIB_IF_ROW2](#) 结构的剩余字段。

请注意，`Netioapi.h` 头文件会自动包含在 `Iphlpapi.h` 头文件中，永远不应直接使用。

要求

最低受支持的客户端	Windows 10 版本 1703 [仅限桌面应用]
最低受支持的服务器	Windows Server 2016 [仅限桌面应用]
目标平台	Windows
标头	<code>netioapi.h</code> (包括 <code>Iphlpapi.h</code>)
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

请参阅

[GetIfEntry](#)

[GetIfEntry2](#)

[GetIfTable](#)

[GetIfTable2](#)

[GetIfTable2Ex](#)

[IP 帮助程序函数参考](#)

[MIB_IFROW](#)

[MIB_IFTABLE](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIfStackTable 函数 (netioapi.h)

项目2023/08/25

GetIfStackTable 函数检索网络接口堆栈行条目表，这些条目指定接口堆栈上网络接口的关系。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIfStackTable(
    [out] PMIB_IFSTACK_TABLE *Table
);
```

parameters

[out] Table

指向缓冲区的指针，该缓冲区接收 [MIB_IFSTACK_TABLE](#) 结构中的接口堆栈行条目表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Table</i> 参数中传递 NULL 指针，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存资源不足，无法完成操作。
ERROR_NOT_FOUND	找不到接口堆栈条目。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

GetIfStackTable 函数在 Windows Vista 及更高版本上定义。

The

[GetIfStackTable](#) 函数枚举本地系统上接口堆栈上的物理和逻辑网络接口，并在 [MIB_IFSTACK_TABLE](#) 结构中返回此信息。

接口堆栈条目在 *Table* 参数指向的缓冲区中的[MIB_IFSTACK_TABLE](#)结构中返回。[MIB_IFSTACK_TABLE](#) 结构包含接口堆栈条目计数和每个接口堆栈条目的[MIB_IFSTACK_ROW](#)结构数组。

接口堆栈中的接口之间的关系是，在 [MIB_IFSTACK_ROW](#) 结构的 [HigherLayerInterfaceIndex](#) 成员中具有索引的接口正上方，其索引位于 [MIB_IFSTACK_ROW](#) 结构的 [LowerLayerInterfaceIndex](#) 成员中。

内存由 [GetIfStackTable](#) 函数为此结构中的 [MIB_IFSTACK_TABLE](#) 和 [MIB_IFSTACK_ROW](#) 项分配。当不再需要这些返回的结构时，通过调用 [FreeMibTable](#) 释放内存。

请注意，*Table* 参数指向的返回[MIB_IFSTACK_TABLE](#)结构可能包含 [NumEntries](#) 成员与 [MIB_IFSTACK_TABLE](#) 结构的 [Table](#) 成员中的第一个[MIB_IFSTACK_ROW](#)数组项之间的对齐填充。[MIB_IFSTACK_ROW](#) 数组条目 之间还可能存在对齐的填充。对 [MIB_IFSTACK_ROW](#) 数组条目的任何访问都应假定可能存在填充。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FreeMibTable](#)

[GetIfEntry2](#)

[GetIfTable2](#)

[GetInvertedIfStackTable](#)

[GetIpInterfaceEntry](#)

[InitializeIpInterfaceEntry](#)

[MIB_IFSTACK_ROW](#)

[MIB_IFSTACK_TABLE](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

[MIB_INVERTEDIFSTACK_ROW](#)

[MIB_INVERTEDIFSTACK_TABLE](#)

[MIB_IPINTERFACE_ROW](#)

[NotifyIpInterfaceChange](#)

[SetIpInterfaceEntry](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

getIfTable 函数 (iphlpapi.h)

项目2023/08/24

GetIfTable 函数检索 MIB-II 接口表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetIfTable(
    [out]     PMIB_IFTABLE pIfTable,
    [in, out] PULONG      pdwSize,
    [in]      BOOL        bOrder
);
```

parameters

[out] pIfTable

指向以MIB_IFTABLE结构的形式接收接口表 [的缓冲区的](#) 指针。

[in, out] pdwSize

在输入时，指定 *pIfTable* 参数指向的缓冲区的大小（以字节为单位）。

在输出时，如果缓冲区不够大，无法容纳返回的接口表，则函数会将此参数设置为等于所需的缓冲区大小（以字节为单位）。

[in] bOrder

一个布尔值，指定是否应按接口索引按升序对返回的接口表进行排序。如果此参数为 TRUE，则对表进行排序。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
------	----

ERROR_INSUFFICIENT_BUFFER	<i>pIfTable</i> 参数指向的缓冲区不够大。 所需大小在 <i>pdwSize</i> 参数指向的 DWORD 变量中返回。
ERROR_INVALID_PARAMETER	<i>pdwSize</i> 参数为 NULL，或者 GetIfTable 无法写入 <i>pdwSize</i> 参数指向的内存。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

The

[GetIfTable](#) 函数枚举本地系统上的物理接口，并在 [MIB_IFTABLE](#) 结构中返回此信息。 物理接口包括软件环回接口。

Windows Vista 及更高版本上提供的 [GetIfTable2](#) 和 [GetIfTable2Ex](#) 函数是 [GetIfTable](#) 函数的增强版本，可枚举本地系统上的物理接口和逻辑接口。 逻辑接口包括用于 L2TP、PPTP、PPOE 和其他隧道封装的各种 WAN 微型端口接口。

接口以 *pIfTable* 参数指向的缓冲区中的[MIB_IFTABLE](#)结构返回。[MIB_IFTABLE](#) 结构包含接口计数和每个接口的[MIB_IFROW](#)结构数组。

请注意，*pIfTable* 参数指向的返回[MIB_IFTABLE](#)结构可能包含 *dwNumEntries* 成员与 [MIB_IFTABLE 结构的表](#)成员中的第一个[MIB_IFROW](#)数组条目之间的对齐填充。

[MIB_IFROW 数组条目](#) 之间可能存在对齐的填充。 对 [MIB_IFROW](#) 数组条目的任何访问都应假定可能存在填充。

示例

以下示例检索接口表，并输出表中的条目数以及每个条目上的一些数据。

C++

```
#include <winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "IPHLPAPI.lib")

#include <iphlpapi.h>

#include <stdio.h>
#include <stdlib.h>

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))
```

```

/* Note: could also use malloc() and free() */

int main()
{
    // Declare and initialize variables.

    DWORD dwSize = 0;
    DWORD dwRetVal = 0;

    unsigned int i, j;

    /* variables used for GetIfTable and GetIfEntry */
    MIB_IFTABLE *pIfTable;
    MIB_IFROW *pIfRow;

    // Allocate memory for our pointers.
    pIfTable = (MIB_IFTABLE *) MALLOC(sizeof(MIB_IFTABLE));
    if (pIfTable == NULL) {
        printf("Error allocating memory needed to call GetIfTable\n");
        return 1;
    }
    // Make an initial call to GetIfTable to get the
    // necessary size into dwSize
    dwSize = sizeof(MIB_IFTABLE);
    if (GetIfTable(pIfTable, &dwSize, FALSE) == ERROR_INSUFFICIENT_BUFFER) {
        FREE(pIfTable);
        pIfTable = (MIB_IFTABLE *) MALLOC(dwSize);
        if (pIfTable == NULL) {
            printf("Error allocating memory needed to call GetIfTable\n");
            return 1;
        }
    }
    // Make a second call to GetIfTable to get the actual
    // data we want.
    if ((dwRetVal = GetIfTable(pIfTable, &dwSize, FALSE)) == NO_ERROR) {
        printf("\tNum Entries: %ld\n\n", pIfTable->dwNumEntries);
        for (i = 0; i < pIfTable->dwNumEntries; i++) {
            pIfRow = (MIB_IFROW *) & pIfTable->table[i];
            printf("\tIndex[%d]:\t %ld\n", i, pIfRow->dwIndex);
            printf("\tInterfaceName[%d]:\t %ws", i, pIfRow->wszName);
            printf("\n");
            printf("\tDescription[%d]:\t ", i);
            for (j = 0; j < pIfRow->dwDescrLen; j++)
                printf("%c", pIfRow->bDescr[j]);
            printf("\n");
            printf("\tType[%d]:\t ", i);
            switch (pIfRow->dwType) {
                case IF_TYPE_OTHER:
                    printf("Other\n");
                    break;
                case IF_TYPE_ETHERNET_CSMACD:
                    printf("Ethernet\n");
                    break;
                case IF_TYPE_ISO88025_TOKENRING:

```

```

        printf("Token Ring\n");
        break;
    case IF_TYPE_PPP:
        printf("PPP\n");
        break;
    case IF_TYPE_SOFTWARE_LOOPBACK:
        printf("Software Lookback\n");
        break;
    case IF_TYPE_ATM:
        printf("ATM\n");
        break;
    case IF_TYPE_IEEE80211:
        printf("IEEE 802.11 Wireless\n");
        break;
    case IF_TYPE_TUNNEL:
        printf("Tunnel type encapsulation\n");
        break;
    case IF_TYPE_IEEE1394:
        printf("IEEE 1394 Firewire\n");
        break;
    default:
        printf("Unknown type %ld\n", pIfRow->dwType);
        break;
}
printf("\tMtu[%d]:\t\t %ld\n", i, pIfRow->dwMtu);
printf("\tSpeed[%d]:\t %ld\n", i, pIfRow->dwSpeed);
printf("\tPhysical Addr:\t ");
if (pIfRow->dwPhysAddrLen == 0)
    printf("\n");
for (j = 0; j < pIfRow->dwPhysAddrLen; j++) {
    if (j == (pIfRow->dwPhysAddrLen - 1))
        printf("%.2X\n", (int) pIfRow->bPhysAddr[j]);
    else
        printf("%.2X-", (int) pIfRow->bPhysAddr[j]);
}
printf("\tAdmin Status[%d]:\t %ld\n", i, pIfRow->dwAdminStatus);
printf("\tOper Status[%d]:\t ", i);
switch (pIfRow->dwOperStatus) {
    case IF_OPER_STATUS_NON_OPERATIONAL:
        printf("Non Operational\n");
        break;
    case IF_OPER_STATUS_UNREACHABLE:
        printf("Unreachable\n");
        break;
    case IF_OPER_STATUS_DISCONNECTED:
        printf("Disconnected\n");
        break;
    case IF_OPER_STATUS_CONNECTING:
        printf("Connecting\n");
        break;
    case IF_OPER_STATUS_CONNECTED:
        printf("Connected\n");
        break;
    case IF_OPER_STATUS_OPERATIONAL:
        printf("Operational\n");

```

```

        break;
    default:
        printf("Unknown status %ld\n", pIfRow->dwAdminStatus);
        break;
    }
    printf("\n");
}
} else {
    printf("GetIfTable failed with error: \n", dwRetVal);
    if (pIfTable != NULL) {
        FREE(pIfTable);
        pIfTable = NULL;
    }
    return 1;
// Here you can use FormatMessage to find out why
// it failed.
}
if (pIfTable != NULL) {
    FREE(pIfTable);
    pIfTable = NULL;
}
return 0;
}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIfEntry](#)

[GetIfEntry2](#)

[GetIfTable2](#)

[GetIfTable2Ex](#)

[GetNumberOfInterfaces](#)

[IP 帮助程序函数参考](#)

[MIB_IFROW](#)

[MIB_IFTABLE](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

反馈

此页面是否有帮助?

[是](#)

[否](#)

[在 Microsoft Q&A 获取帮助](#)

getIfTable2 函数 (netioapi.h)

项目2023/08/25

GetIfTable2 函数检索 MIB-II 接口表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIfTable2(
    [out] PMIB_IF_TABLE2 *Table
);
```

parameters

[out] Table

指向缓冲区的指针，该缓冲区接收 [MIB_IF_TABLE2](#) 结构中的接口表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_NOT_ENOUGH_MEMORY	内存资源不足，无法完成操作。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

The

GetIfTable2 函数枚举本地系统上的逻辑和物理接口，并在 [MIB_IF_TABLE2](#) 结构中返回此信息。GetIfTable2 是 GetIfTable 函数的增强版本。

类似的 [GetIfTable2Ex](#) 函数可用于指定要返回的接口级别。调用 *Level* 参数设置为 [MibIfTableNormal](#) 的 GetIfTable2Ex 函数将检索与调用 GetIfTable2 函数相同的结果。

接口在 *Table* 参数指向的缓冲区中的 MIB_IF_TABLE2 结构中返回。 MIB_IF_TABLE2 结构包含每个接口的接口计数和 MIB_IF_ROW2 结构的数组。 内存由 GetIfTable2 函数为此结构中的 MIB_IF_TABLE2 和 MIB_IF_ROW2 项分配。 当不再需要这些返回的结构时，通过调用 FreeMibTable 释放内存。

请注意，*Table* 参数指向的返回 MIB_IF_TABLE2 结构可能包含 NumEntries 成员与 MIB_IF_TABLE2 结构的 Table 成员中的第一个 MIB_IF_ROW2 数组条目之间的对齐填充。 MIB_IF_ROW2 数组条目 之间还可能存在对齐的填充。 对 MIB_IF_ROW2 数组条目的任何访问都应假定可能存在填充。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FreeMibTable](#)

[GetIfTable](#)

[GetIfTable2Ex](#)

[IP 帮助程序函数参考](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获取帮助

getIfTable2Ex 函数 (netioapi.h)

项目2023/08/25

GetIfTable2Ex 函数检索 MIB-II 接口表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIfTable2Ex(
    [in]  MIB_IF_TABLE_LEVEL Level,
    [out] PMIB_IF_TABLE2      *Table
);
```

parameters

[in] Level

要检索的接口信息的级别。此参数可以是 *Netioapi.h* 头文件中定义的 **MIB_IF_TABLE_LEVEL** 枚举类型的值之一。

值	含义
MibIfTableNormal	指定此参数时, <i>table</i> 参数指向的 MIB_IF_TABLE2 结构中的 MIB_IF_ROW2 结构的成员中返回的统计信息和状态值将从筛选器堆栈的顶部返回。
MibIfTableRaw	<i>Table</i> 参数指向的 MIB_IF_TABLE2 结构中 MIB_IF_ROW2结构的成员 中返回的统计信息和状态值将直接返回要查询的接口。

[out] Table

指向缓冲区的指针, 该缓冲区接收 **MIB_IF_TABLE2** 结构中的接口表。

返回值

如果函数成功, 则返回值NO_ERROR。

如果函数失败, 则返回值为以下错误代码之一。

返回代码	说明
------	----

ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Level</i> 参数中传递了非法值，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	可用内存资源不足，无法完成该操作。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

The

[GetIfTable2Ex](#) 函数枚举本地系统上的逻辑接口和物理接口，并在 [MIB_IF_TABLE2](#) 结构中返回此信息。[GetIfTable2Ex](#) 是 [GetIfTable](#) 函数的增强版本，允许选择要检索的接口信息的级别。

类似的 [GetIfTable2](#) 函数也可用于检索接口。但不允许指定要返回的接口级别。调用 *Level* 参数设置为 [MibIfTableNormal](#) 的 [GetIfTable2Ex](#) 函数将检索与调用 [GetIfTable2](#) 函数相同的结果。

接口在 *Table* 参数指向的缓冲区中的[MIB_IF_TABLE2](#)结构中返回。[MIB_IF_TABLE2](#) 结构包含每个接口的接口计数和[MIB_IF_ROW2](#)结构数组。[内存由 GetIfTable2](#) 函数为此结构中的 [MIB_IF_TABLE2](#) 和 [MIB_IF_ROW2](#) 项分配。如果不再需要这些返回的结构，请通过调用 [FreeMibTable](#) 释放内存。

对于 *Level* 参数的任一可能值，将返回包括 NDIS 中间驱动程序接口和 NDIS 筛选器驱动程序接口在内的所有接口。*Level* 参数的设置会影响接口的 *Table*参数指向的 [MIB_IF_TABLE2](#) 结构中[MIB_IF_ROW2](#) 结构的统计信息和状态成员的返回方式。例如，卡 (NIC) 的网络接口将具有 NDIS 微型端口驱动程序。可以安装 NDIS 中间驱动程序，以在上层协议驱动程序和 NDIS 微型端口驱动程序之间进行接口。可以在 NDIS 中间驱动程序的顶部附加 NDIS 筛选器驱动程序 (LWF)。假设 NIC 将 [MIB_IF_ROW2](#) 结构的 [MediaConnectState](#) 成员报告为 [MediaConnectStateConnected](#)，但 NDIS 筛选器驱动程序修改状态并将状态报告为 [MediaConnectStateDisconnected](#)。在将 *Level* 参数设置为 [MibIfTableNormal](#) 的情况下查询接口信息时，将报告筛选器堆栈顶部的状态（即 [MediaConnectStateDisconnected](#)）。查询接口时，将 *Level* 参数设置为 [MibIfTableRaw](#) 时，将返回接口级别的直接状态，即 [MediaConnectStateConnected](#)。

请注意，*Table* 参数指向的返回[MIB_IF_TABLE2](#)结构可能包含 [NumEntries](#) 成员与 [MIB_IF_TABLE2](#) 结构的 *Table* 成员中第一个[MIB_IF_ROW2](#) 数组条目之间的对齐填充。[MIB_IF_ROW2](#) 数组条目 之间也可能存在对齐填充。对 [MIB_IF_ROW2](#) 数组条目的任何访问都应假定可能存在填充。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FreeMibTable](#)

[GetIfTable](#)

[GetIfTable2](#)

[IP 帮助程序函数参考](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getInterfaceActiveTimestampCapabilitie s 函数 (iphlpapi.h)

项目2023/08/24

检索网络适配器当前启用的时间戳功能。

有关详细信息和代码示例，请参阅[数据包时间戳](#)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetInterfaceActiveTimestampCapabilities(  
    const NET_LUID           *InterfaceLuid,  
    PINTERFACE_TIMESTAMP_CAPABILITIES TimestampCapabilites  
) ;
```

parameters

InterfaceLuid

类型: `_In_ CONST NET_LUID*`

网络本地唯一标识符 (要为其检索当前启用时间戳功能的网络适配器的 LUID)。

TimestampCapabilites

类型: `_Out_ PINTERFACE_TIMESTAMP_CAPABILITIES`

如果函数成功，`TimestampCapabilites` 将返回描述当前启用的时间戳功能的结构。

返回值

类型: `DWORD`

如果该函数成功，则返回 `NO_ERROR`。如果对应于 `InterfaceLuid` 的网络卡无法识别时间戳，则该函数将返回 `ERROR_NOT_SUPPORTED`。如果网络卡驱动程序颁发不受支持的时间戳配置，则该函数将返回 `ERROR_BAD_DRIVER`。

要求

最低受支持的客户端	Windows 10内部版本 20348
最低受支持的服务器	Windows 10内部版本 20348
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

- [包时间戳](#)

反馈

此页面是否有帮助？



[在 Microsoft Q&A 获取帮助](#)

getInterfaceDnsSettings 函数 (netioapi.h)

项目2023/04/22

从 *Interface* 参数中指定的接口检索 DNS 设置。完成返回的设置对象后，必须调用 [FreeInterfaceDnsSettings](#) 以释放它。

语法

C++

```
NETIOAPI_API GetInterfaceDnsSettings(  
    GUID             Interface,  
    DNS_INTERFACE_SETTINGS *Settings  
) ;
```

parameters

`Interface`

类型: `_In_ GUID`

设置引用的 COM 接口的 GUID。

`Settings`

类型: `_Inout_ const DNS_INTERFACE_SETTINGS*`

`GetInterfaceDnsSettings` 填充此结构中的所有设置。

应仅设置 *Version* 成员; 标志字段必须为空。

如果将 *Version* 成员设置为 `DNS_INTERFACE_SETTINGS_VERSION1`，则 *Settings* 参数必须指向有效的 `DNS_INTERFACE_SETTINGS` 结构。

如果将 *Version* 成员设置为 `DNS_INTERFACE_SETTINGS_VERSION2`，则 *Settings* 参数必须指向有效的 `DNS_INTERFACE_SETTINGS_EX` 结构。

如果将 *Version* 成员设置为 `DNS_INTERFACE_SETTINGS_VERSION3`，则 *Settings* 参数必须指向有效的 `DNS_INTERFACE_SETTINGS3` 结构。

返回值

如果成功，则返回NO_ERROR。非零返回值指示失败。

要求

最低受支持的客户端	Windows 10内部版本 19041
最低受支持的服务器	Windows 10内部版本 19041
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

- [SetInterfaceDnsSettings](#)
- [FreeInterfaceDnsSettings](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getInterfaceInfo 函数 (iphlpapi.h)

项目2023/08/24

GetInterfaceInfo 函数获取本地系统上启用了 IPv4 的网络接口适配器的列表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetInterfaceInfo(
    [out]      PIP_INTERFACE_INFO pIfTable,
    [in, out]   PULONG          dwOutBufLen
);
```

parameters

[out] pIfTable

指向缓冲区的指针，该缓冲区指定接收适配器列表的 [IP_INTERFACE_INFO](#) 结构。此缓冲区必须由调用方分配。

[in, out] dwOutBufLen

指向 [DWORD](#) 变量的指针，该变量指定 *pIfTable* 参数指向的缓冲区的大小，以接收 [IP_INTERFACE_INFO](#) 结构。如果此大小不足以保存 IPv4 接口信息，则 [GetInterfaceInfo](#) 会用所需的大小填充此变量，并返回 [ERROR_INSUFFICIENT_BUFFER](#) 错误代码。

返回值

如果函数成功，则返回值 [NO_ERROR](#)。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	用于接收 IPv4 适配器信息的缓冲区太小。如果 <i>dwOutBufLen</i> 参数指示 <i>pIfTable</i> 参数指向的缓冲区太小，无法检索 IPv4 接口信息，则返回此值。所需大小在 <i>dwOutBufLen</i> 参数指向的 DWORD 变量中返回。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>dwOutBufLen</i> 参数为 NULL ，或者 GetInterfaceInfo 无法写入 <i>dwOutBufLen</i> 参数

	指向的内存，则返回此错误。
ERROR_NO_DATA	本地系统上没有为 IPv4 启用网络适配器。如果禁用本地系统上的所有网络适配器，也会返回此值。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`GetInterfaceInfo` 函数特定于启用了 IPv4 的网络适配器。该函数返回 *pIfTable* 参数指向的 [IP_INTERFACE_INFO](#) 结构，该结构包含本地系统上启用了 IPv4 的网络适配器的数量，以及一组 [IP_ADAPTER_INDEX_MAP](#) 结构，其中包含每个启用了 IPv4 的网络适配器上的信息。`GetInterfaceInfo` 返回的 [IP_INTERFACE_INFO](#) 结构至少包含一个 [IP_ADAPTER_INDEX_MAP](#) 结构，即使 [IP_INTERFACE_INFO](#) 结构的 *NumAdapters* 成员指示未启用具有 IPv4 的网络适配器。当 `GetInterfaceInfo` 返回的 [IP_INTERFACE_INFO](#) 结构的 *NumAdapters* 成员为零时，[IP_INTERFACE_INFO](#) 结构中返回的单个 [IP_ADAPTER_INDEX_MAP](#) 结构的成员的值是不确定的。

如果调用 `GetInterfaceInfo` 函数的缓冲区太小，无法检索 IPv4 接口信息 (*dwOutBufLen* 参数指示 *pIfTable* 参数指向的缓冲区太小)，则函数返回 [ERROR_INSUFFICIENT_BUFFER](#)。所需大小在 *dwOutBufLen* 参数指向的 [DWORD](#) 变量中返回。

使用 `GetInterfaceInfo` 函数的正确方法是调用此函数两次。在第一次调用中，在 *pIfTable* 参数中传递 `NULL` 指针，在 *dwOutBufLen* 参数指向的变量中传递零。调用将失败并返回 [ERROR_INSUFFICIENT_BUFFER](#)，并且 *dwOutBufLen* 参数指向的 [DWORD](#) 变量中返回此缓冲区的所需大小。然后，可以使用 *dwOutBufLen* 指向的值分配所需大小的缓冲区。然后，可以使用 *pIfTable* 参数中传递的指向此缓冲区的指针以及设置为此缓冲区大小的缓冲区长度，再次调用 `GetInterfaceInfo` 函数。

`GetAdaptersInfo` 和 `GetInterfaceInfo` 函数不返回有关环回接口的信息。有关环回接口的信息由 `GetIpAddrTable` 函数返回。

在 Windows Vista 及更高版本中，[IP_INTERFACE_INFO](#) 结构中返回的 [IP_ADAPTER_INDEX_MAP](#) 结构的 *Name* 成员可能是网络接口的 GUID 的 Unicode 字符串，(字符串以“{”字符) 开头。

示例

以下示例检索本地系统上启用了 IPv4 的网络适配器的列表，并打印第一个网络适配器的各种属性。

C++

```
#include <winsock2.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int main()
{

    // Declare and initialize variables
    PIP_INTERFACE_INFO pInfo = NULL;
    ULONG ulOutBufLen = 0;

    DWORD dwRetVal = 0;
    int iReturn = 1;

    int i;

    // Make an initial call to GetInterfaceInfo to get
    // the necessary size in the ulOutBufLen variable
    dwRetVal = GetInterfaceInfo(NULL, &ulOutBufLen);
    if (dwRetVal == ERROR_INSUFFICIENT_BUFFER) {
        pInfo = (IP_INTERFACE_INFO *) MALLOC(ulOutBufLen);
        if (pInfo == NULL) {
            printf
                ("Unable to allocate memory needed to call
GetInterfaceInfo\n");
            return 1;
        }
    }
    // Make a second call to GetInterfaceInfo to get
    // the actual data we need
    dwRetVal = GetInterfaceInfo(pInfo, &ulOutBufLen);
    if (dwRetVal == NO_ERROR) {
        printf("Number of Adapters: %ld\n\n", pInfo->NumAdapters);
        for (i = 0; i < pInfo->NumAdapters; i++) {
            printf("Adapter Index[%d]: %ld\n", i,
                pInfo->Adapter[i].Index);
            printf("Adapter Name[%d]: %ws\n\n", i,
                pInfo->Adapter[i].Name);
        }
        iReturn = 0;
    } else if (dwRetVal == ERROR_NO_DATA) {
        printf
            ("There are no network adapters with IPv4 enabled on the local
system\n");
    }
}
```

```
iReturn = 0;
} else {
    printf("GetInterfaceInfo failed with error: %d\n", dwRetVal);
    iReturn = 1;
}

FREE(pInfo);
return (iReturn);
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetAdaptersInfo](#)

[GetIpAddrTable](#)

[GetNumberOfInterfaces](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IP_ADAPTER_INDEX_MAP](#)

[IP_INTERFACE_INFO](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

getInterfaceSupportedTimestampCapabilities 函数 (iphlpapi.h)

项目2023/08/24

检索网络适配器支持的时间戳功能。

有关详细信息和代码示例，请参阅[数据包时间戳](#)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetInterfaceSupportedTimestampCapabilities(
    const NET_LUID           *InterfaceLuid,
    PINTERFACE_TIMESTAMP_CAPABILITIES TimestampCapabilites
);
```

parameters

InterfaceLuid

类型: `_In_ CONST NET_LUID*`

网络本地唯一标识符 (要为其检索支持的时间戳功能的网络适配器的 LUID)。

TimestampCapabilites

类型: `_Out_ PINTERFACE_TIMESTAMP_CAPABILITIES`

如果函数成功，`TimestampCapabilites` 将返回描述支持的时间戳功能的结构。

返回值

类型: `DWORD`

如果该函数成功，则返回 `NO_ERROR`。如果对应于 `InterfaceLuid` 的网络卡无法识别时间戳，则该函数将返回 `ERROR_NOT_SUPPORTED`。

要求

最低受支持的客户端	Windows 10内部版本 20348
最低受支持的服务器	Windows 10内部版本 20348
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

- [包时间戳](#)

反馈

此页面是否有帮助？



[在 Microsoft Q&A 获取帮助](#)

getInvertedIfStackTable 函数 (netioapi.h)

项目2023/08/25

GetInvertedIfStackTable 函数检索反转网络接口堆栈行条目的表，这些条目指定接口堆栈上网络接口的关系。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetInvertedIfStackTable(
    [out] PMIB_INVERTEDIFSTACK_TABLE *Table
);
```

parameters

[out] Table

指向缓冲区的指针，该缓冲区接收 MIB_INVERTEDIFSTACK_TABLE 结构中反转接口堆栈行项的表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Table 参数中传递 NULL 指针，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存资源不足，无法完成操作。
ERROR_NOT_FOUND	找不到接口堆栈条目。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

`GetInvertedIfStackTable` 函数是在 Windows Vista 及更高版本上定义的。

The

`GetInvertedIfStackTable` 函数枚举本地系统上接口堆栈上的物理和逻辑网络接口，并在 [MIB_INVERTEDIFSTACK_TABLE](#) 结构中以反转形式返回此信息。

接口堆栈条目在 *Table* 参数指向的缓冲区中的[MIB_INVERTEDIFSTACK_TABLE](#)结构中返回。[MIB_INVERTEDIFSTACK_TABLE](#) 结构包含接口堆栈条目计数和每个接口堆栈条目的[MIB_INVERTEDIFSTACK_ROW](#)结构数组。

接口堆栈中的接口之间的关系是，在 [MIB_INVERTEDIFSTACK_ROW](#) 结构的 `HigherLayerInterfaceIndex` 成员中具有索引的接口紧邻[MIB_INVERTEDIFSTACK_ROW](#)结构的`LowerLayerInterfaceIndex` 成员中具有索引的接口上方。

内存由 `GetInvertedIfStackTable` 函数为此结构中的 [MIB_INVERTEDIFSTACK_TABLE](#) 结构和 [MIB_INVERTEDIFSTACK_ROW](#) 项分配。当不再需要这些返回的结构时，通过调用 [FreeMibTable](#) 释放内存。

请注意，*Table* 参数指向的返回[MIB_INVERTEDIFSTACK_TABLE](#)结构可能包含 `NumEntries` 成员与 [MIB_INVERTEDIFSTACK_TABLE](#) 结构的Table 成员中的第一个[MIB_INVERTEDIFSTACK_ROW](#)数组条目之间的对齐填充。[MIB_INVERTEDIFSTACK_ROW](#) 数组条目之间也可能存在对齐的填充。对 [MIB_INVERTEDIFSTACK_ROW](#) 数组条目的任何访问都应假定可能存在填充。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FreeMibTable](#)

[GetIfEntry2](#)

[GetIfStackTable](#)

[GetIfTable2](#)

[GetIpInterfaceEntry](#)

[InitializeIpInterfaceEntry](#)

[MIB_IFSTACK_ROW](#)

[MIB_IFSTACK_TABLE](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

[MIB_INVERTEDIFSTACK_ROW](#)

[MIB_INVERTEDIFSTACK_TABLE](#)

[MIB_IPINTERFACE_ROW](#)

[NotifyIpInterfaceChange](#)

[SetIpInterfaceEntry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getIpAddrTable 函数 (iphlpapi.h)

项目2023/08/24

GetIpAddrTable 函数检索接口到 IPv4 地址映射表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetIpAddrTable(
    [out]      PMIB_IPADDRTABLE pIpAddrTable,
    [in, out]   PULONG          pdwSize,
    [in]        BOOL           bOrder
);
```

parameters

[out] pIpAddrTable

指向缓冲区的指针，该缓冲区接收接口到 IPv4 地址映射表作为 [MIB_IPADDRTABLE](#) 结构。

[in, out] pdwSize

输入时，指定 *pIpAddrTable* 参数指向的缓冲区的大小（以字节为单位）。

在输出时，如果缓冲区不够大，无法容纳返回的映射表，则函数会将此参数设置为等于所需的缓冲区大小（以字节为单位）。

[in] bOrder

如果此参数为 TRUE，则返回的映射表将按 IPv4 地址的升序排序。排序按网络字节顺序执行。例如，10.0.0.255 紧接在 10.0.1.0 之前。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码

说明

ERROR_INSUFFICIENT_BUFFER	<i>pIpAddrTable</i> 参数指向的缓冲区不够大。 所需大小在 <i>pdwSize</i> 参数指向的 DWORD 变量中返回。
ERROR_INVALID_PARAMETER	<i>pdwSize</i> 参数为 NULL, 或者 GetIpAddrTable 无法写入 <i>pdwSize</i> 参数指向的内存。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetIpAddrTable](#) 函数在本地计算机上检索接口到 IPv4 地址映射表，并在 [MIB_IPADDRTABLE](#) 结构中返回此信息。

[GetIpAddrTable](#) 函数返回的 IPv4 地址受本地计算机上网络接口的状态的影响。 手动重置网络接口卡 (NIC) 和某些 PnP 事件可能会导致 IP 地址被删除或更改。

在 Windows Server 2003 和 Windows XP 上，如果通过调用 [DisableMediaSense](#) 函数禁用了本地计算机上 TCP/IP 堆栈的媒体感知功能，则 [GetIpAddrTable](#) 函数返回的 IPv4 地址也会受到影响。 禁用媒体感知后，[GetIpAddrTable](#) 函数可能会返回与断开连接的接口关联的 IPv4 地址。 断开连接的接口的这些 IPv4 地址无效。

在 Windows Server 2008 和 Windows Vista 上，[GetIpAddrTable](#) 函数返回的 IPv4 地址不受本地计算机上 TCP/IP 堆栈的媒体感知功能的影响。[GetIpAddrTable](#) 函数仅返回有效的 IPv4 地址。

Windows XP 上可用的 [GetAdaptersAddresses](#) 函数可用于检索 IPv6 和 IPv4 地址和接口信息。

[GetIpAddrTable](#) 函数返回的 [MIB_IPADDRTABLE](#) 结构可能包含用于在 *dwNumEntries* 成员与表成员中的第一个 [MIB_IPADDRROW](#) 数组条目之间进行对齐的填充。 表成员中的 [MIB_IPADDRROW](#) 数组条目之间也可能存在对齐填充。 对 [MIB_IPADDRROW](#) 数组条目的任何访问都应假定可能存在填充。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，[MIB_IPADDRROW](#) 在 *lpmib.h* 头文件中定义，而不是在 *lprtrmib.h* 头文件中定义。 请注意，*lpmib.h* 头文件会自动包含在 *lphlpapi.h* 头文件中的 *lprtrmib.h* 中。 不应直接使用 *lpmib.h* 和 *lprtrmib.h* 头文件。

示例

以下示例检索 IP 地址表，然后打印表中 IP 地址条目的一些成员。

C++

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int __cdecl main()
{
    int i;

    /* Variables used by GetIpAddrTable */
    PMIB_IPADDRTABLE pIPAddrTable;
    DWORD dwSize = 0;
    DWORD dwRetVal = 0;
    IN_ADDR IPAddr;

    /* Variables used to return error message */
    LPVOID lpMsgBuf;

    // Before calling AddIPAddress we use GetIpAddrTable to get
    // an adapter to which we can add the IP.
    pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(sizeof(MIB_IPADDRTABLE));

    if (pIPAddrTable) {
        // Make an initial call to GetIpAddrTable to get the
        // necessary size into the dwSize variable
        if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==
            ERROR_INSUFFICIENT_BUFFER) {
            FREE(pIPAddrTable);
            pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(dwSize);

        }
        if (pIPAddrTable == NULL) {
            printf("Memory allocation failed for GetIpAddrTable\n");
            exit(1);
        }
    }
    // Make a second call to GetIpAddrTable to get the
    // actual data we want
    if ( (dwRetVal = GetIpAddrTable( pIPAddrTable, &dwSize, 0 )) != NO_ERROR
) {
        printf("GetIpAddrTable failed with error %d\n", dwRetVal);
        if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL,
dwRetVal,
```

```

MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),           // Default language
                                                (LPTSTR) & lpMsgBuf, 0, NULL)) {
    printf("\tError: %s", lpMsgBuf);
    LocalFree(lpMsgBuf);
}
exit(1);
}

printf("\tNum Entries: %ld\n", pIPAddrTable->dwNumEntries);
for (i=0; i < (int) pIPAddrTable->dwNumEntries; i++) {
    printf("\n\tInterface Index[%d]:\t%ld\n", i, pIPAddrTable-
>table[i].dwIndex);
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwAddr;
    printf("\tIP Address[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwMask;
    printf("\tSubnet Mask[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwBCastAddr;
    printf("\tBroadCast[%d]:\t%s (%ld)\n", i, inet_ntoa(IPAddr),
pIPAddrTable->table[i].dwBCastAddr);
    printf("\tReassembly size[%d]:\t%ld\n", i, pIPAddrTable-
>table[i].dwReasmSize);
    printf("\tType and State[%d]:", i);
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_PRIMARY)
        printf("\tPrimary IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DYNAMIC)
        printf("\tDynamic IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DISCONNECTED)
        printf("\tAddress is on disconnected interface");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DELETED)
        printf("\tAddress is being deleted");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_TRANSIENT)
        printf("\tTransient address");
    printf("\n");
}

if (pIPAddrTable) {
    FREE(pIPAddrTable);
    pIPAddrTable = NULL;
}

exit(0);
}

```

要求

最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[AddIPAddress](#)

[DisableMediaSense](#)

[GetAdaptersAddresses](#)

[IP 帮助程序函数参考](#)

[IP_ADAPTER_ADDRESSES](#)

[MIB_IPADDRROW](#)

[MIB_IPADDRTABLE](#)

[RestoreMediaSense](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIpErrorString 函数 (iphlpapi.h)

项目2023/08/24

GetIpErrorString 函数检索 IP 帮助程序错误字符串。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetIpErrorString(
    [in]      IP_STATUS ErrorCode,
    [out]     PWSTR      Buffer,
    [in, out] PDWORD     Size
);
```

parameters

[in] ErrorCode

要检索的错误代码。此参数的可能值在 *Ipexport.h* 头文件中定义。

[out] Buffer

指向包含错误代码字符串的缓冲区的指针，如果函数返回并NO_ERROR。

[in, out] Size

指向 DWORD 的指针，该指针指定 *Buffer* 参数指向的缓冲区的长度（以字符为单位），不包括终止 null (即缓冲区的大小（以字符为单位），减去一）。

返回值

成功后返回NO_ERROR。

如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

注解

GetIpErrorString 函数可用于检索 IP 错误代码的 IP 帮助程序错误字符串。在 *ErrorCode* 参数中传递的IP_STATUS错误代码在 ICMP 和 ICMPv6 函数使用的[ICMP_ECHO_REPLY](#)、[ICMP_ECHO_REPLY32](#)和[ICMPV6_ECHO_REPLY](#)结构的[状态](#)成员中返回。使用这些结构的

函数包括 [Icmp6ParseReplies](#)、[Icmp6SendEcho2](#)、[IcmpParseReplies](#)、[IcmpSendEcho](#)、[IcmpSendEcho2](#) 和 [IcmpSendEcho2Ex](#)。

在 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上, [GetIpErrorMessage](#) 函数的语法略有更改。 *Buffer* 参数的数据类型已从 **PWCHAR** 更改为 **PWSTR**。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[ICMPV6_ECHO_REPLY](#)

[ICMP_ECHO_REPLY](#)

[ICMP_ECHO_REPLY32](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[Icmp6ParseReplies](#)

[Icmp6Sendecho2](#)

[IcmpParseReplies](#)

[IcmpSendecho](#)

[IcmpSendecho2](#)

[Icmpsendecho2Ex](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIpForwardEntry2 函数 (netioapi.h)

项目2023/08/25

GetIpForwardEntry2 函数检索本地计算机上的 IP 路由条目的信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIpForwardEntry2(
    [in, out] PMIB_IPFORWARD_ROW2 Row
);
```

parameters

[in, out] Row

指向 IP 路由条目 [MIB_IPFORWARD_ROW2](#) 结构条目的指针。 成功返回后，将使用 IP 路由条目的属性更新此结构。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果在 Row 参数中传递 NULL 指针，未指定 Row 参数所指向 MIB_IPFORWARD_ROW2 的 DestinationPrefix 成员、Row 参数指向 MIB_IPFORWARD_ROW2 的 NextHop 成员，或者 Row 参数指向的 MIB_IPFORWARD_ROW2 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误未指定 参数。
ERROR_NOT_FOUND	找不到元素。 如果 Row 参数指向的 MIB_IPFORWARD_ROW2 结构的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口与 MIB_IPFORWARD_ROW2 结构的 DestinationPrefix 成员中指定的 IP 地址前缀和地址系列不匹配，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。 如果本地计算机上没有 IPv4 堆栈，并且已在 Row 参数指向的 MIB_IPFORWARD_ROW2 的

`DestinationPrefix` 成员的地址系列中指定了 `AF_INET`，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且为 `DestinationPrefix` 成员中的地址系列指定了 `AF_INET6`，也会返回此错误。

其他

使用 [FormatMessage](#) 获取返回错误的消息字符串。

注解

`GetIpForwardEntry2` 函数在 Windows Vista 及更高版本上定义。

`GetIpForwardEntry2` 函数用于检索 [MIB_IPFORWARD_ROW2](#) 结构条目。

输入时，`Row` 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构中的 `DestinationPrefix` 成员必须初始化为有效的 IPv4 或 IPv6 地址前缀和系列。输入时，`Row` 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构中的 `NextHop` 成员必须初始化为有效的 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 `Row` 参数的 [MIB_IPFORWARD_ROW2](#) 结构中的至少一个成员：`InterfaceLuid` 或 `InterfaceIndex`。

字段按上面列出的顺序使用。因此，如果指定了 `InterfaceLuid`，则使用此成员来确定接口。如果未为 `InterfaceLuid` 成员设置值（此成员的值设置为零），则接下来使用 `InterfaceIndex` 成员来确定接口。

在调用成功时输出时，`GetIpForwardEntry2` 检索 IP 路由条目的其他属性，并填写 `Row` 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构。

`Row` 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构的 `Metric` 成员中指定的路由指标偏移量仅代表整个路由指标的一部分。完整指标是添加到关联接口的 [MIB_IPINTERFACE_ROW](#) 结构的 `Metric` 成员中指定的接口指标的路由指标的组合。应用程序可以通过调用 `GetInterfaceEntry` 函数来检索接口指标。

可以调用 `GetIpForwardTable2` 函数来枚举本地计算机上的 IP 路由条目。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)

Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpForwardEntry2](#)

[DeleteIpForwardEntry2](#)

[GetBestRoute2](#)

[GetIpForwardTable2](#)

[GetIpInterfaceEntry](#)

[InitializeIpForwardEntry](#)

[MIB_IPFORWARD_ROW2](#)

[MIB_IPFORWARD_TABLE2](#)

[MIB_IPINTERFACE_ROW](#)

[NotifyRouteChange2](#)

[SetIpForwardEntry2](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIpForwardTable 函数 (iphlpapi.h)

项目2023/08/24

GetIpForwardTable 函数检索 IPv4 路由表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetIpForwardTable(
    [out]      PMIB_IPFORWARDTABLE pIpForwardTable,
    [in, out]   PULONG          pdwSize,
    [in]        BOOL           bOrder
);
```

parameters

[out] pIpForwardTable

指向以MIB_IPFORWARDTABLE结构的形式接收 IPv4 路由表 的缓冲区的 指针。

[in, out] pdwSize

输入时，指定 *pIpForwardTable* 参数指向的缓冲区的大小（以字节为单位）。

在输出时，如果缓冲区的大小不足以容纳返回的路由表，则函数会将此参数设置为等于所需的缓冲区大小（以字节为单位）。

[in] bOrder

一个布尔值，指定是否应对返回的表进行排序。如果此参数为 TRUE，则按以下顺序对表进行排序：

1. 目标地址
2. 生成路由的协议
3. 多路径路由策略
4. 下一跃点地址

返回值

如果函数成功，则返回值为 零) (NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	<i>pIpForwardTable</i> 参数指向的缓冲区不够大。 所需大小在 <i>pdwSize</i> 参数指向的 DWORD 变量中返回。
ERROR_INVALID_PARAMETER	<i>pdwSize</i> 参数为 NULL，或者 GetIpForwardTable 无法写入 <i>pdwSize</i> 参数指向的内存。
ERROR_NO_DATA	没有可用的数据。 如果本地计算机上没有路由，则返回此错误。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。 如果本地计算机上未安装任何 IP 堆栈，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[MIB_IPFORWARDROW](#) 结构的 *dwForwardProto* 成员指定生成路由的协议或路由机制。有关可能的协议和路由机制的列表，请参阅协议 [标识符](#)。

[MIB_IPFORWARDROW](#) 结构的 *dwForwardDest*、*dwForwardMask* 和 *dwForwardNextHop* 成员以网络字节顺序表示 IPv4 地址。

[MIB_IPFORWARDROW](#) 结构的 *dwForwardDest* 成员中 0.0.0.0 的 IPv4 地址被视为默认路由。如果安装了多个网络适配器，[MIB_IPFORWARDTABLE](#) 可能包含多个 [MIB_IPFORWARDROW](#) 条目，其中 *dwForwardDest* 成员设置为 0.0.0.0。

当 *dwForwardAge* 设置为 **INFINITE** 时，不会根据超时删除路由值。 *dwForwardAge* 的任何其他值指定在网络路由表中添加或修改路由后的秒数。

在 Windows Server 2003 或 Windows 2000 Server 上，当路由和远程访问服务 (RRAS) 运行时，返回的[MIB_IPFORWARDROW](#) 条目将 *dwForwardType* 和 *dwForwardAge* 成员设置为零。

在 Windows Vista 和 Windows Server 2008 上，在 [MIB_IPFORWARDROW](#) 结构的 *dwForwardMetric1* 成员中指定的路由指标表示添加到关联接口的 [MIB_INTERFACE_ROW](#) 结构的**指标**成员中指定的接口指标的组合。因此，[MIB_IPFORWARDROW](#) 结构的 *dwForwardMetric1* 成员应等于或大于关联 [MIB_INTERFACE_ROW](#) 结构的 **Metric** 成员。如果应用程序想要在 Windows Vista 和 Windows Server 2008 上将路由指标设置为 0，则应将 [MIB_IPFORWARDROW](#) 结构的 *dwForwardMetric1* 成员设置为等于关联 [MIB_INTERFACE_ROW](#) 结构的**指标**成员中指定的接口**指标**的值。应用程序可以通过调用 [GetInterfaceEntry](#) 函数来检索接口指标。

IPv4 路由当前不使用 GetIpForwardTable 返回的MIB_IPFORWARDROW结构条目的一些成员。这些成员包括 dwForwardPolicy、dwForwardNextHopAS、dwForwardMetric2、dwForwardMetric3、dwForwardMetric4 和 dwForwardMetric5。

示例

以下示例检索 IP 路由表，然后打印表中每个路由的一些字段。

C++

```
// Need to link with Ws2_32.lib and Iphlpapi.lib
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int main()
{
    // Declare and initialize variables.

    /* variables used for GetIpForwardTable */
    PMIB_IPFORWARDTABLE pIpForwardTable;
    DWORD dwSize = 0;
    DWORD dwRetVal = 0;

    char szDestIp[128];
    char szMaskIp[128];
    char szGatewayIp[128];

    struct in_addr IpAddr;

    int i;

    pIpForwardTable =
        (MIB_IPFORWARDTABLE *) MALLOC(sizeof(MIB_IPFORWARDTABLE));
    if (pIpForwardTable == NULL) {
        printf("Error allocating memory\n");
        return 1;
    }

    if (GetIpForwardTable(pIpForwardTable, &dwSize, 0) ==
```

```

        ERROR_INSUFFICIENT_BUFFER) {
    FREE(pIpForwardTable);
    pIpForwardTable = (MIB_IPFORWARDTABLE *) MALLOC(dwSize);
    if (pIpForwardTable == NULL) {
        printf("Error allocating memory\n");
        return 1;
    }
}

/* Note that the IPv4 addresses returned in
 * GetIpForwardTable entries are in network byte order
 */
if ((dwRetVal = GetIpForwardTable(pIpForwardTable, &dwSize, 0)) ==
NO_ERROR) {
    printf("\tNumber of entries: %d\n",
           (int) pIpForwardTable->dwNumEntries);
    for (i = 0; i < (int) pIpForwardTable->dwNumEntries; i++) {
        /* Convert IPv4 addresses to strings */
        IpAddr.S_un.S_addr =
            (u_long) pIpForwardTable->table[i].dwForwardDest;
        strcpy_s(szDestIp, sizeof (szDestIp), inet_ntoa(IpAddr));
        IpAddr.S_un.S_addr =
            (u_long) pIpForwardTable->table[i].dwForwardMask;
        strcpy_s(szMaskIp, sizeof (szMaskIp), inet_ntoa(IpAddr));
        IpAddr.S_un.S_addr =
            (u_long) pIpForwardTable->table[i].dwForwardNextHop;
        strcpy_s(szGatewayIp, sizeof (szGatewayIp), inet_ntoa(IpAddr));

        printf("\n\tRoute[%d] Dest IP: %s\n", i, szDestIp);
        printf("\tRoute[%d] Subnet Mask: %s\n", i, szMaskIp);
        printf("\tRoute[%d] Next Hop: %s\n", i, szGatewayIp);
        printf("\tRoute[%d] If Index: %ld\n", i,
               pIpForwardTable->table[i].dwForwardIfIndex);
        printf("\tRoute[%d] Type: %ld - ", i,
               pIpForwardTable->table[i].dwForwardType);
        switch (pIpForwardTable->table[i].dwForwardType) {
        case MIB_IPROUTE_TYPE_OTHER:
            printf("other\n");
            break;
        case MIB_IPROUTE_TYPE_INVALID:
            printf("invalid route\n");
            break;
        case MIB_IPROUTE_TYPE_DIRECT:
            printf("local route where next hop is final destination\n");
            break;
        case MIB_IPROUTE_TYPE_INDIRECT:
            printf
                ("remote route where next hop is not final
destination\n");
            break;
        default:
            printf("UNKNOWN Type value\n");
            break;
        }
        printf("\tRoute[%d] Proto: %ld - ", i,

```

```

        pIpForwardTable->table[i].dwForwardProto);
    switch (pIpForwardTable->table[i].dwForwardProto) {
    case MIB IPPROTO_OTHER:
        printf("other\n");
        break;
    case MIB IPPROTO_LOCAL:
        printf("local interface\n");
        break;
    case MIB IPPROTO_NETMGMT:
        printf("static route set through network management \n");
        break;
    case MIB IPPROTO_ICMP:
        printf("result of ICMP redirect\n");
        break;
    case MIB IPPROTO_EGP:
        printf("Exterior Gateway Protocol (EGP)\n");
        break;
    case MIB IPPROTO_GGP:
        printf("Gateway-to-Gateway Protocol (GGP)\n");
        break;
    case MIB IPPROTO_HELLO:
        printf("Hello protocol\n");
        break;
    case MIB IPPROTO_RIP:
        printf("Routing Information Protocol (RIP)\n");
        break;
    case MIB IPPROTO_IS_IS:
        printf
            ("Intermediate System-to-Intermediate System (IS-IS)
protocol\n");
        break;
    case MIB IPPROTO_ES_IS:
        printf("End System-to-Intermediate System (ES-IS)
protocol\n");
        break;
    case MIB IPPROTO_CISCO:
        printf("Cisco Interior Gateway Routing Protocol (IGRP)\n");
        break;
    case MIB IPPROTO_BBN:
        printf("BBN Internet Gateway Protocol (IGP) using SPF\n");
        break;
    case MIB IPPROTO_OSPF:
        printf("Open Shortest Path First (OSPF) protocol\n");
        break;
    case MIB IPPROTO_BGP:
        printf("Border Gateway Protocol (BGP)\n");
        break;
    case MIB IPPROTO_NT_AUTOSTATIC:
        printf("special Windows auto static route\n");
        break;
    case MIB IPPROTO_NT_STATIC:
        printf("special Windows static route\n");
        break;
    case MIB IPPROTO_NT_STATIC_NON_DOD:
        printf

```

```

        ("special Windows static route not based on Internet
standards\n");
        break;
    default:
        printf("UNKNOWN Proto value\n");
        break;
}

printf("\tRoute[%d] Age: %ld\n", i,
       pIpForwardTable->table[i].dwForwardAge);
printf("\tRoute[%d] Metric1: %ld\n", i,
       pIpForwardTable->table[i].dwForwardMetric1);
}
FREE(pIpForwardTable);
return 0;
} else {
    printf("\tGetIpForwardTable failed.\n");
    FREE(pIpForwardTable);
    return 1;
}

}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[CreateIpForwardEntry](#)

[DeleteIpForwardEntry](#)

[GetIpInterfaceEntry](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPFORWARDTABLE](#)

[MIB_IPINTERFACE_ROW](#)

协议标识符

[SetIpForwardEntry](#)

反馈

此页面是否有帮助？

[!\[\]\(ed4f2118f1594beef3df6b9647246493_img.jpg\) 是](#)

[!\[\]\(81b2bb5552938b7257e3af45752798f9_img.jpg\) 否](#)

[在 Microsoft Q&A 获得帮助](#)

getIpForwardTable2 函数 (netioapi.h)

项目2023/08/25

GetIpForwardTable2 函数检索本地计算机上的 IP 路由条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIpForwardTable2(
    [in] ADDRESS_FAMILY Family,
    [out] PMIB_IPFORWARD_TABLE2 *Table
);
```

parameters

[in] Family

要检索的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和 PF_ 协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织已更改，并且 Ws2def.h 头文件中定义了此成员的可能值。请注意，Ws2def.h 头文件自动包含在 Winsock2.h 中，不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	未指定地址系列。指定此参数后，此函数返回包含 IPv4 和 IPv6 条目的 IP 路由表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数时，此函数返回仅包含 IPv4 条目的 IP 路由表。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数时，此函数返回仅包含 IPv6 条目的 IP 路由表。

[out] Table

指向 [MIB_IPFORWARD_TABLE2](#) 结构的指针，该结构包含本地计算机上的 IP 路由条目表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Table</i> 参数中传递 NULL 指针，或者未将 <i>Family</i> 参数指定为 AF_INET、AF_INET6 或 AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	可用内存资源不足，无法完成该操作。
ERROR_NOT_FOUND	未找到 <i>Family</i> 参数中指定的 IP 路由条目。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Family 参数中指定了AF_INET ，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 Family 参数中指定了AF_INET6 ，也会返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetIpForwardTable2](#) 函数在 Windows Vista 及更高版本上定义。

The

[GetIpForwardTable2](#) 函数枚举本地系统上的 IP 路由条目，并在 [MIB_IPFORWARD_TABLE2](#) 结构中返回此信息。

IP 路由条目在 *Table* 参数指向的缓冲区中的[MIB_IPFORWARD_TABLE2](#)结构中返回。[MIB_IPFORWARD_TABLE2](#) 结构包含每个 IP 路由条目的 IP 路由条目计数和 [MIB_IPFORWARD_ROW2](#)结构数组。如果不再需要这些返回的结构，请通过调用 [FreeMibTable](#) 释放内存。

Family 参数必须初始化为AF_INET、AF_INET6或AF_UNSPEC。

请注意，*Table* 参数指向的返回[MIB_IPFORWARD_TABLE2](#)结构可能包含 NumEntries 成员与 [MIB_IPFORWARD_TABLE2](#) 结构的Table 成员中第一个[MIB_IPFORWARD_ROW2](#)数组条

目之间的对齐填充。 MIB_IPFORWARD_ROW2 数组条目 之间也可能存在对齐的填充。 对 MIB_IPFORWARD_ROW2 数组条目的任何访问都应假定可能存在填充。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpForwardEntry2](#)

[DeleteIpForwardEntry2](#)

[FreeMibTable](#)

[GetBestRoute2](#)

[GetIpForwardEntry2](#)

[InitializeIpForwardEntry](#)

[MIB_IPFORWARD_ROW2](#)

[MIB_IPFORWARD_TABLE2](#)

[NotifyRouteChange2](#)

[SetIpForwardEntry2](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

getIpInterfaceEntry 函数 (netioapi.h)

项目2023/08/25

GetIpInterfaceEntry 函数检索本地计算机上指定接口的 IP 信息。

语法

C++

```
IPHLAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIpInterfaceEntry(
    [in, out] PMIB_IPINTERFACE_ROW Row
);
```

parameters

[in, out] Row

指向 [MIB_IPINTERFACE_ROW](#) 结构的指针，该结构在成功返回时接收本地计算机上的接口的信息。输入时，**必须将**[MIB_IPINTERFACE_ROW](#)的 InterfaceLuid 或 InterfaceIndex 成员设置为要为其检索信息的接口。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果由 Row 参数指向的 MIB_IPINTERFACE_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数所指向 MIB_IPINTERFACE_ROW 的 Family 成员未指定为 AF_INET 或 AF_INET6，或者 Row 参数指向的 MIB_IPINTERFACE_ROW 的 InterfaceLuid 或 InterfaceIndex 成员均未指定，则返回此错误。
ERROR_NOT_FOUND	找不到元素。如果由 Row 参数指向的 MIB_IPINTERFACE_ROW 结构的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口与

MIB_IPINTERFACE_ROW 结构中的 Family 成员中指定的 IP 地址系列不匹配，则返回此错误。

其他

使用 [FormatMessage](#) 函数获取返回错误的消息字符串。

注解

[GetIpInterfaceEntry](#) 函数在 Windows Vista 及更高版本上定义。

输入时，Row 参数指向的 MIB_IPINTERFACE_ROW 结构中的 Family 成员必须初始化为 AF_INET 或 AF_INET6。此外，在输入时，必须初始化指向 Row 参数的 MIB_IPINTERFACE_ROW 结构中的至少一个成员：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则使用此成员来确定接口。如果未为 InterfaceLuid 成员设置值（此成员的值设置为零），则接下来使用 InterfaceIndex 成员来确定接口。

在输出中，如果指定了 InterfaceIndex，则填充 Row 参数指向的 MIB_IPINTERFACE_ROW 结构的 InterfaceLuid 成员。Row 参数指向的 MIB_IPINTERFACE_ROW 结构的其他成员也将填充。

[InitializeIpInterfaceEntry](#) 函数必须用于使用默认值初始化 MIB_IPINTERFACE_ROW 结构条目的字段。然后，应用程序可以更改要修改的 MIB_IPINTERFACE_ROW 条目中的字段，然后调用 [SetIpInterfaceEntry](#) 函数。

无特权同时访问具有不同安全要求的多个网络会产生安全漏洞，并允许无特权应用程序意外地在两个网络之间中继数据。典型示例是同时访问虚拟专用网络（VPN）和 Internet。Windows Server 2003 和 Windows XP 使用弱主机模型，其中 RAS 通过增加所有默认路由的路由指标（超过其他接口）来阻止此类同时访问。因此，所有流量都通过 VPN 接口路由，从而中断其他网络连接。

在 Windows Vista 及更高版本上，默认使用强主机模型。如果使用 [GetBestRoute2](#) 或 [GetBestRoute](#) 在路由查找中指定了源 IP 地址，则路由查找仅限于源 IP 地址的接口。RAS 的路由指标修改不起作用，因为潜在路由列表甚至没有 VPN 接口的路由，从而允许流量发往 Internet。MIB_IPINTERFACE_ROW 的 DisableDefaultRoutes 成员可用于在接口上使用默认路由禁用。VPN 客户端可以将此成员用作安全措施，以在 VPN 客户端不需要拆分隧道时限制拆分隧道。VPN 客户端可以调用 [SetIpInterfaceEntry](#) 函数，以便在需要时将 DisableDefaultRoutes 成员设置为 TRUE。VPN 客户端可以通过调用 [GetIpInterfaceEntry](#) 函数查询 DisableDefaultRoutes 成员的当前状态。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[GetBestRoute](#)

[GetBestRoute2](#)

[GetIfEntry2](#)

[GetIfTable2](#)

[GetIfTable2Ex](#)

[GetIpInterfaceTable](#)

[IP 帮助程序函数参考](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

[MIB_IPINTERFACE_ROW](#)

[MIB_IPINTERFACE_TABLE](#)

[SetIpInterfaceEntry](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

getIpInterfaceTable 函数 (netioapi.h)

项目2023/08/25

GetIpInterfaceTable 函数检索本地计算机上的 IP 接口条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIpInterfaceTable(
    [in] ADDRESS_FAMILY           Family,
    [out] PMIB_IPINTERFACE_TABLE *Table
);
```

parameters

[in] Family

要检索的 IP 接口的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和 PF_ 协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在 Windows Vista 和更高版本以及 Windows SDK 上，头文件的组织已更改，此成员的可能值在 *Ws2def.h* 头文件中定义。请注意，*Ws2def.h* 头文件会自动包含在 *Winsock2.h* 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	地址系列未指定。指定此参数后， GetIpInterfaceTable 函数将返回包含 IPv4 和 IPv6 条目的 IP 接口表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。

[out] Table

指向缓冲区的指针，该缓冲区接收 **MIB_IPINTERFACE_TABLE** 结构中的 IP 接口条目表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Table</i> 参数中传递 NULL 指针，或者 <i>Family</i> 参数未指定为AF_INET、AF_INET6或AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存资源不足，无法完成操作。
ERROR_NOT_FOUND	未找到 <i>Family</i> 参数中指定的 IP 接口条目。
ERROR_NOT_SUPPORTED	不支持 函数。如果未在本地计算机上配置 <i>Address</i> 参数中指定的 IP 传输，则返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

`GetIpInterfaceTable` 函数在 Windows Vista 及更高版本上定义。

The

`GetIpInterfaceTable` 函数枚举本地系统上的 IP 接口，并在 [MIB_IPINTERFACE_TABLE](#) 结构中返回此信息。

IP 接口条目在 *Table* 参数指向的缓冲区中的[MIB_IPINTERFACE_TABLE](#)结构中返回。

[MIB_IPINTERFACE_TABLE](#)结构包含 IP 接口条目计数和每个 IP 接口条目的

[MIB_IPINTERFACE_ROW](#)结构数组。当不再需要这些返回的结构时，通过调用 [FreeMibTable](#) 释放内存。

Family 参数必须初始化为 AF_INET 或 AF_INET6。

请注意，*Table* 参数指向的返回[MIB_IPINTERFACE_TABLE](#)结构可能包含 *NumEntries* 成员与 [MIB_IPINTERFACE_TABLE](#) 结构的*Table* 成员中的第一个[MIB_IPINTERFACE_ROW](#)数组条目之间的对齐填充。[MIB_IPINTERFACE_ROW](#)数组条目之间也可能存在对齐的填充。对 [MIB_IPINTERFACE_ROW](#) 数组条目的任何访问都应假定可能存在填充。

示例

以下示例检索 IP 接口表，然后打印表中 IP 接口条目的几个成员的值。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>
#include <winsock2.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "iphlpapi.lib")

int main()
{
    // Declare and initialize variables

    int i;

    DWORD dwRetVal = 0;

    PMIB_IPINTERFACE_TABLE pipTable = NULL;

    dwRetVal = GetIpInterfaceTable(AF_UNSPEC, &pipTable);
    if (dwRetVal != NO_ERROR) {
        printf("GetIpInterfaceTable returned error: %d\n", dwRetVal);
        exit(1);
    }
    // Print some variables from the rows in the table
    printf("Number of table entries: %d\n\n", pipTable->NumEntries);

    for (i = 0; i < (int) pipTable->NumEntries; i++) {
        printf("Address Family[%d]:\t\t", i);
        switch (pipTable->Table[i].Family) {
        case AF_INET:
            printf("IPv4\n");
            break;
        case AF_INET6:
            printf("IPv6\n");
            break;
        default:
            printf("Other: %d\n", pipTable->Table[i].Family);
            break;
        }

        printf("Interface LUID NetLuidIndex[%d]:\t %lu\n",
               i, pipTable->Table[i].InterfaceLuid.Info.NetLuidIndex);

        printf("Interface LUID IfType[%d]:\t ", i);
        switch (pipTable->Table[i].InterfaceLuid.Info.IfType) {
        case IF_TYPE_OTHER:
            printf("Other\n");
            break;
        }
    }
}
```

```

        break;
    case IF_TYPE_ETHERNET_CSMACD:
        printf("Ethernet\n");
        break;
    case IF_TYPE_IS088025_TOKENRING:
        printf("Token ring\n");
        break;
    case IF_TYPE_PPP:
        printf("PPP\n");
        break;
    case IF_TYPE_SOFTWARE_LOOPBACK:
        printf("Software loopback\n");
        break;
    case IF_TYPE_ATM:
        printf("ATM\n");
        break;
    case IF_TYPE_IEEE80211:
        printf("802.11 wireless\n");
        break;
    case IF_TYPE_TUNNEL:
        printf("Tunnel encapsulation\n");
        break;
    case IF_TYPE_IEEE1394:
        printf("IEEE 1394 (Firewire)\n");
        break;
    default:
        printf("Unknown: %d\n",
               pipTable->Table[i].InterfaceLuid.Info.IfType);
        break;
    }
    printf("Interface Index[%d]:\t\t %lu\n",
           i, pipTable->Table[i].InterfaceIndex);
    printf("Maximum reassembly size[%d]:\t %lu\n", i,
           pipTable->Table[i].MaxReassemblySize);

    printf("Advertising enabled[%d]:\t\t ", i);
    if (pipTable->Table[i].AdvertisingEnabled)
        printf("Yes\n");
    else
        printf("No\n");

    printf("Forwarding enabled[%d]:\t\t ", i);
    if (pipTable->Table[i].ForwardingEnabled)
        printf("Yes\n");
    else
        printf("No\n");

    printf("Network layer MTU[%d]:\t\t %lu\n", i, pipTable-
>Table[i].NlMtu);

    printf("Connected[%d]:\t\t\t ", i);
    if (pipTable->Table[i].Connected)
        printf("Yes\n");
    else
        printf("No\n");

```

```

printf("Supports wakeup patterns[%d]:\t ", i);
if (pipTable->Table[i].SupportsWakeUpPatterns)
    printf("Yes\n");
else
    printf("No\n");

printf("Supports neighbor discovery[%d]:\t ", i);
if (pipTable->Table[i].SupportsNeighborDiscovery)
    printf("Yes\n");
else
    printf("No\n");

printf("Supports router discovery[%d]:\t ", i);
if (pipTable->Table[i].SupportsRouterDiscovery)
    printf("Yes\n");
else
    printf("No\n");

printf("\n");
}

FreeMibTable(pipTable);
pipTable = NULL;

exit(0);
}

```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FreeMibTable](#)

[GetIfEntry2](#)

[GetIfStackTable](#)

[GetIfTable2](#)

[GetInvertedIfStackTable](#)

[GetIpInterfaceEntry](#)

[IP 帮助程序函数参考](#)

[InitializeIpInterfaceEntry](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

[MIB_IPINTERFACE_ROW](#)

[MIB_IPINTERFACE_TABLE](#)

[NotifyIpInterfaceChange](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIpNetEntry2 函数 (netioapi.h)

项目2023/08/25

GetIpNetEntry2 函数检索本地计算机上邻居 IP 地址条目的信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIpNetEntry2(
    [in, out] PMIB_IPNET_ROW2 Row
);
```

parameters

[in, out] Row

指向邻居 IP 地址条目 [MIB_IPNET_ROW2](#) 结构条目的指针。成功返回后，将使用邻居 IP 地址的属性更新此结构。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果由 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数指向 MIB_IPNET_ROW2 的 Address 成员未设置为有效的邻居 IPv4 或 IPv6 地址，或者未指定 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误。
ERROR_NOT_FOUND	找不到元素。如果 Row 参数指向的 MIB_IPNET_ROW2 结构的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口与 MIB_IPNET_ROW2 结构中的Address 成员中指定的邻居 IP 地址和地址系列不匹配，则返回此错误。

ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Row 参数指向的 MIB_IPNET_ROW2 结构的 Address 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且已在 MIB_IPNET_ROW2 结构的 Address 成员中指定了 IPv6 地址，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`GetIpNetEntry2` 函数在 Windows Vista 及更高版本上定义。

`GetIpNetEntry2` 函数用于检索[MIB_IPNET_ROW2](#)结构条目。

输入时，Row 参数指向的 [MIB_IPNET_ROW2](#) 结构中的 Address 成员必须初始化为有效的邻居 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 Row 参数的 [MIB_IPNET_ROW2](#) 结构中的至少一个成员：`InterfaceLuid` 或 `InterfaceIndex`。

字段按上面列出的顺序使用。因此，如果指定了 `InterfaceLuid`，则使用此成员来确定接口。如果未为 `InterfaceLuid` 成员设置值（此成员的值设置为零），则接下来使用 `InterfaceIndex` 成员来确定接口。

调用成功时，`GetIpNetEntry2` 在输出中检索邻居 IP 地址的其他属性，并填写 Row 参数指向的 [MIB_IPNET_ROW2](#) 结构。

可以调用 [GetIpNetTable2](#) 函数来枚举本地计算机上的邻居 IP 地址条目。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpNetEntry2](#)

[DeleteIpNetEntry2](#)

[FlushIpNetTable2](#)

[GetIpNetTable2](#)

[MIB_IPNET_ROW2](#)

[MIB_IPNET_TABLE2](#)

[ResolveIpNetEntry2](#)

[SetIpNetEntry2](#)

反馈

此页面是否有帮助？

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

getIpNetTable 函数 (iphlpapi.h)

项目2023/08/24

GetIpNetTable 函数检索 IPv4 到物理地址的映射表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetIpNetTable(
    [out]     PMIB_IPNETTABLE IpNetTable,
    [in, out] PULONG          SizePointer,
    [in]      BOOL           Order
);
```

parameters

[out] IpNetTable

指向缓冲区的指针，该缓冲区接收 IPv4 到物理地址映射表作为 MIB_IPNETTABLE 结构。

[in, out] SizePointer

在输入时，指定 *pIpNetTable* 参数指向的缓冲区的大小（以字节为单位）。

在输出时，如果缓冲区不够大，无法容纳返回的映射表，则函数会将此参数设置为等于所需的缓冲区大小（以字节为单位）。

[in] Order

一个布尔值，指定是否应按 IP 地址按升序对返回的映射表进行排序。如果此参数为 TRUE，则对表进行排序。

返回值

如果函数成功，则返回值NO_ERROR或ERROR_NO_DATA。

如果函数失败或未返回任何数据，则返回值为以下错误代码之一。

返回代码

说明

ERROR_INSUFFICIENT_BUFFER	<i>pIpNetTable</i> 参数指向的缓冲区不够大。 所需大小在 <i>pdwSize</i> 参数指向的 DWORD 变量中返回。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果 <i>pdwSize</i> 参数为 NULL, 或者 GetIpNetTable 无法写入 <i>pdwSize</i> 参数指向的内存, 则返回此错误。
ERROR_NO_DATA	没有要返回的数据。 IPv4 到物理地址的映射表为空。 此返回值指示对 GetIpNetTable 函数的调用成功, 但没有要返回的数据。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

The

[GetIpNetTable](#) 函数枚举本地系统上从 IPv4 到物理地址映射表的 IPv4 的地址解析协议 (ARP) 条目, 并在 [MIB_IPNETTABLE](#) 结构中返回此信息。

IPv4 地址条目以 *pIpNetTable* 参数指向的缓冲区中的[MIB_IPNETTABLE](#)结构返回。
[MIB_IPNETTABLE](#) 结构包含每个 IPv4 地址条目的 ARP 条目计数和[MIB_IPNETROW](#)结构数组。

请注意, *pIpNetTable* 参数指向的返回[MIB_IPNETTABLE](#)结构可能包含**用于在 dwNumEntries 成员与 MIB_IPNETTABLE 结构的表成员中的第一个 MIB_IPNETROW 数组条目之间进行对齐的填充**。 **MIB_IPNETROW 数组条目** 之间也可能存在对齐填充。 对 **MIB_IPNETROW 数组条目** 的任何访问都应假定可能存在填充。

在 Windows Vista 及更高版本中, [GetIpNetTable2](#) 函数可用于检索 IPv6 和 IPv4 的邻居 IP 地址。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h

Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateIpNetEntry](#)

[DeleteIpNetEntry](#)

[FlushIpNetTable](#)

[GetIpNetTable2](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPNETROW](#)

[MIB_IPNETTABLE](#)

[SetIpNetEntry](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIpNetTable2 函数 (netioapi.h)

项目2023/08/25

GetIpNetTable2 函数检索本地计算机上的 IP 邻居表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIpNetTable2(
    [in] ADDRESS_FAMILY Family,
    [out] PMIB_IPNET_TABLE2 *Table
);
```

parameters

[in] Family

要检索的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和 PF_ 协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 *Ws2def.h* 头文件中定义了此成员的可能值。请注意，*Ws2def.h* 头文件会自动包含在 *Winsock2.h* 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	地址系列未指定。指定此参数时，此函数返回包含 IPv4 和 IPv6 条目的邻居 IP 地址表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数时，此函数返回仅包含 IPv4 条目的邻居 IP 地址表。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数时，此函数返回仅包含 IPv6 条目的邻居 IP 地址表。

[out] Table

指向 MIB_IPNET_TABLE2 结构的指针，该结构包含本地计算机上的邻居 IP 地址条目表。

返回值

如果函数成功，则返回值为NO_ERROR或ERROR_NOT_FOUND。

如果函数失败或未返回任何数据，则返回值是以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Table</i> 参数中传递 NULL 指针，或者 <i>Family</i> 参数未指定为AF_INET、AF_INET6或AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存资源不足，无法完成操作。
ERROR_NOT_FOUND	未找到 <i>Family</i> 参数中指定的邻居 IP 地址条目。 此返回值表示对 GetIpNetTable2 函数的调用成功，但没有要返回的数据。在 <i>Family</i> 参数中指定了AF_INET并且没有要返回的 ARP 条目时，可能会发生这种情况。
ERROR_NOT_SUPPORTED	不支持该请求。 如果本地计算机上没有 IPv4 堆栈，并且已在 <i>Family</i> 参数中指定了AF_INET，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且已在 <i>Family</i> 参数中指定了AF_INET6，则也会返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

GetIpNetTable2 函数在 Windows Vista 及更高版本上定义。

The

GetIpNetTable2 函数枚举本地系统上的相邻 IP 地址，并在 [MIB_IPNET_TABLE2](#) 结构中返回此信息。

邻居 IP 地址条目在 *Table* 参数指向的缓冲区中的[MIB_IPNET_TABLE2](#)结构中返回。

[MIB_IPNET_TABLE2](#)结构包含相邻 IP 地址条目计数和每个相邻 IP 地址条目的[MIB_IPNET_ROW2](#)结构数组。当不再需要这些返回的结构时，通过调用 [FreeMibTable](#) 释放内存。

Family 参数必须初始化为 AF_INET、AF_INET6 或 AF_UNSPEC。

请注意，*Table* 参数指向的返回[MIB_IPNET_TABLE2](#)结构可能包含 NumEntries 成员与 [MIB_IPNET_TABLE2](#) 结构的Table 成员中第一个[MIB_IPNET_ROW2](#)数组条目之间的对齐填

充。 MIB_IPNET_ROW2 数组条目 之间还可能存在对齐的填充。 对 MIB_IPNET_ROW2 数组条目的任何访问都应假定可能存在填充。

示例

以下示例检索 IP 邻居表，然后打印表中 IP 邻居行条目的值。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>
#include <winsock2.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

int main()
{
    // Declare and initialize variables

    int i;
    unsigned int j;
    unsigned long status = 0;

    PMIB_IPNET_TABLE2 pipTable = NULL;
//    MIB_IPNET_ROW2 ipRow;

    status = GetIpNetTable2(AF_INET, &pipTable);
    if (status != NO_ERROR) {
        printf("GetIpNetTable for IPv4 table returned error: %ld\n",
    status);
        exit(1);
    }
    // Print some variables from the table
    printf("Number of IPv4 table entries: %d\n\n", pipTable->NumEntries);

    for (i = 0; (unsigned) i < pipTable->NumEntries; i++) {
//        printf("Table entry: %d\n", i);
        printf("IPv4 Address[%d]:\t %s\n", (int) i,
            inet_ntoa(pipTable->Table[i].Address.Ipv4.sin_addr));
        printf("Interface index[%d]:\t\t %lu\n", (int) i,
            pipTable->Table[i].InterfaceIndex);

        printf("Interface LUID NetLuidIndex[%d]:\t %lu\n",
            (int) i,
```

```

        (int) i, pipTable->Table[i].InterfaceLuid.Info.NetLuidIndex);
printf("Interface LUID IfType[%d]: ", (int) i);
switch (pipTable->Table[i].InterfaceLuid.Info.IfType) {
    case IF_TYPE_OTHER:
        printf("Other\n");
        break;
    case IF_TYPE_ETHERNET_CSMACD:
        printf("Ethernet\n");
        break;
    case IF_TYPE_ISO88025_TOKENRING:
        printf("Token ring\n");
        break;
    case IF_TYPE_PPP:
        printf("PPP\n");
        break;
    case IF_TYPE_SOFTWARE_LOOPBACK:
        printf("Software loopback\n");
        break;
    case IF_TYPE_ATM:
        printf("ATM\n");
        break;
    case IF_TYPE_IEEE80211:
        printf("802.11 wireless\n");
        break;
    case IF_TYPE_TUNNEL:
        printf("Tunnel encapsulation\n");
        break;
    case IF_TYPE_IEEE1394:
        printf("IEEE 1394 (Firewire)\n");
        break;
    default:
        printf("Unknown: %d\n",
               pipTable->Table[i].InterfaceLuid.Info.IfType);
        break;
}

printf("Physical Address[%d]:\t ", (int) i);
if (pipTable->Table[i].PhysicalAddressLength == 0)
    printf("\n");
//    for (j = 0; (unsigned) j < pipTable-
>Table[i].PhysicalAddressLength; j++)
//        printf ("%c"
for (j = 0; j < pipTable->Table[i].PhysicalAddressLength; j++) {
    if (j == (pipTable->Table[i].PhysicalAddressLength - 1))
        printf("%.2X\n", (int) pipTable-
>Table[i].PhysicalAddress[j]);
    else
        printf("%.2X-", (int) pipTable-
>Table[i].PhysicalAddress[j]);
}

printf("Physical Address Length[%d]:\t %lu\n", (int) i,
       pipTable->Table[i].PhysicalAddressLength);

printf("Neighbor State[%d]:\t ", (int) i);

```

```

        switch (pipTable->Table[i].State) {
            case NlnsUnreachable:
                printf("NlnsUnreachable\n");
                break;
            case NlnsIncomplete:
                printf("NlnsIncomplete\n");
                break;
            case NlnsProbe:
                printf("NlnsProbe\n");
                break;
            case NlnsDelay:
                printf("NlnsDelay\n");
                break;
            case NlnsStale:
                printf("NlnsStale\n");
                break;
            case NlnsReachable:
                printf("NlnsReachable\n");
                break;
            case NlnsPermanent:
                printf("NlnsPermanent\n");
                break;
            default:
                printf("Unknown: %d\n", pipTable->Table[i].State);
                break;
        }

        printf("Flags[%d]:\t\t %u\n", (int) i,
               (unsigned char) pipTable->Table[i].Flags);

        printf("ReachabilityTime[%d]:\t %lu\n\n", (int) i,
               pipTable->Table[i].ReachabilityTime);

    }
    FreeMibTable(pipTable);
    pipTable = NULL;

    exit(0);
}

```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows

标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpNetEntry2](#)

[FlushIpNetTable2](#)

[FreeMibTable](#)

[GetIpNetEntry2](#)

[MIB_IPNET_ROW2](#)

[MIB_IPNET_TABLE2](#)

[ResolveIpNetEntry2](#)

[SetIpNetEntry2](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIpNetworkConnectionBandwidthEstimates 函数 (netioapi.h)

项目2023/08/25

GetIpNetworkConnectionBandwidthEstimates 函数检索指定接口上网络连接的历史带宽估计值。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetIpNetworkConnectionBandwidthEstimates(
    [in]    NET_IFINDEX                           InterfaceIndex,
    [in]    ADDRESS_FAMILY                        AddressFamily,
    [out]   PMIB_IP_NETWORK_CONNECTION_BANDWIDTH_ESTIMATES BandwidthEstimates
);
```

parameters

[in] InterfaceIndex

网络接口的本地索引值。

当禁用然后启用网络适配器时，或者在其他情况下，此索引值可能会更改，不应被视为永久性。

[in] AddressFamily

地址系列。 *Ws2def.h* 头文件中列出了地址系列的可能值。 请注意，AF_ 地址系列和PF_ 协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

请注意，*Ws2def.h* 头文件自动包含在 *Winsock2.h* 中，不应直接使用。

当前支持的值是 AF_INET 或 AF_INET6，即 IPv4 和 IPv6 的 Internet 地址系列格式。

值	含义
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。
AF_INET6	Internet 协议版本 6 (IPv6) 地址系列。

[out] BandwidthEstimates

指向缓冲区的指针，该缓冲区返回为接口当前连接到的附件点保留的历史带宽估计值。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果 <i>InterfaceIndex</i> 参数指定的接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>BandwidthEstimates</i> 参数中传递 NULL 指针，或者 <i>AddressFamily</i> 参数未指定为 AF_INET 或 AF_INET6，则返回此错误。
ERROR_NOT_FOUND	找不到元素。如果 <i>InterfaceIndex</i> 参数指定的网络接口与 <i>AddressFamily</i> 参数中指定的 IP 地址系列不匹配，则返回此错误。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

GetIpNetworkConnectionBandwidthEstimates 函数在 Windows 8 及更高版本上定义。

输入时，*AddressFamily* 参数必须初始化为 AF_INET 或 AF_INET6。此外，在输入时，必须使用指定的接口索引初始化 *InterfaceIndex* 参数。

必须为 *InterfaceIndex* 参数设置值（此参数的值不得设置为零）。

在[输出时](#)，如果指定了 *AddressFamily* 和 *InterfaceIndex* 参数，则填充 *BandwidthEstimates* 参数指向的 MIB_IP_NETWORK_CONNECTION_BANDWIDTH_ESTIMATES 结构。

GetIpNetworkConnectionBandwidthEstimates 函数返回附件（第一个跃点）供应用程序使用的可用带宽的历史估计值。这些估计值旨在作为优化性能参数的指南，应用程序应维护阈值并区分低带宽和高带宽情况的行为。

随着在同一网络上竞争的设备消耗更多带宽，真正的可用带宽可能会随时间而改变。因此，应用程序应准备好处理可用带宽低于 `GetIpNetworkConnectionBandwidthEstimates` 函数报告的历史限制的情况。

TCP/IP 堆栈可能未在特定或双向方向上为给定接口生成任何估计值。在这种情况下，返回的估计值为零。应用程序应准备好通过选择合理的默认值和微调（如果需要）来处理此类情况。

`Netioapi.h` 头文件自动包含在 `Iphlpapi.h` 头文件中。永远不应直接使用 `Netioapi.h` 头文件。

要求

最低受支持的客户端	Windows 8 [仅限桌面应用]
最低受支持的服务器	Windows Server 2012 [仅限桌面应用]
目标平台	Windows
标头	<code>netioapi.h</code> (包括 <code>Iphlpapi.h</code>)
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[MIB_IP_NETWORK_CONNECTION_BANDWIDTH_ESTIMATES](#)

[NL_BANDWIDTH_INFORMATION](#)

[TCP_ESTATS_BANDWIDTH_ROD_v0](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

getIpPathEntry 函数 (netioapi.h)

项目2023/08/25

GetIpPathEntry 函数检索本地计算机上的 IP 路径条目的信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIpPathEntry(
    [in, out] PMIB_IPPATH_ROW Row
);
```

parameters

[in, out] Row

指向 IP 路径条目 MIB_IPPATH_ROW 结构条目的指针。 成功返回后，将使用 IP 路径条目的属性更新此结构。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。 如果由 Row 参数指向的 MIB_IPPATH_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	参数不正确。 如果在 Row 参数中传递 NULL 指针，Row 参数指向的MIB_IPPATH_ROW的 Destination 成员中的 si_family 成员未设置为 AF_INET 或 AF_INET6，或者未指定 Row 参数指向的MIB_IPPATH_ROW的 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误。 如果 Row 参数指向 MIB_IPPATH_ROW 的 Source 成员中的 si_family 成员与目标 IP 地址系列不匹配，并且源 IP 地址 si_family 未指定为 AF_UNSPEC，则也会返回此错误。

ERROR_NOT_FOUND	找不到元素。如果由 <i>Row</i> 参数指向的 MIB_IPPATH_ROW 结构的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口与 MIB_IPPATH_ROW 结构中 Destination 成员中指定的 IP 地址和地址系列不匹配，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 <i>Row</i> 参数指向的 MIB_IPPATH_ROW 的 Source 和 Destination 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 源 成员和 目标 成员中指定了 IPv6 地址，也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetIpPathEntry](#) 函数在 Windows Vista 及更高版本上定义。

[GetIpPathEntry](#) 函数用于检索[MIB_IPPATH_ROW](#)结构条目。

输入时，*Row* 参数指向的 [MIB_IPPATH_ROW](#) 结构中的 **Destination** 成员必须初始化为有效的 IPv4 或 IPv6 地址和系列。[MIB_IPPATH_ROW](#)结构中的**源**成员中指定的地址系列还必须与**目标**成员中指定的目标 IP 地址系列匹配，或者**源**成员中的地址系列必须指定为 **AF_UNSPEC**。此外，必须初始化指向 *Row* 参数的 [MIB_IPPATH_ROW](#) 结构中的至少一个成员：**InterfaceLuid** 或 **InterfaceIndex**。

字段按上面列出的顺序使用。因此，如果指定了 **InterfaceLuid**，则使用此成员来确定接口。如果没有为 **InterfaceLuid** 成员设置值（此成员的值设置为零），则接下来使用 **InterfaceIndex** 成员来确定接口。

在调用成功时输出时，[GetIpPathEntry](#) 检索 IP 路径条目的其他属性，并填充 *Row* 参数指向的[MIB_IPPATH_ROW](#)结构。

可以调用 [GetIpPathTable](#) 函数来枚举本地计算机上的 IP 路径条目。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)

Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FlushIpPathTable](#)

[GetIpPathTable](#)

[MIB_IPPATH_ROW](#)

[MIB_IPPATH_TABLE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIpPathTable 函数 (netioapi.h)

项目2023/08/25

GetIpPathTable 函数检索本地计算机上的 IP 路径表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetIpPathTable(
    [in] ADDRESS_FAMILY Family,
    [out] PMIB_IPPATH_TABLE *Table
);
```

parameters

[in] Family

要检索的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和PF_ 协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织已更改，并且 *Ws2def.h* 头文件中定义了此成员的可能值。请注意，*Ws2def.h* 头文件自动包含在 *Winsock2.h* 中，不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	未指定地址系列。指定此参数时，此函数返回包含 IPv4 和 IPv6 条目的 IP 路径表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数后，此函数返回仅包含 IPv4 条目的 IP 路径表。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数时，此函数返回仅包含 IPv6 条目的 IP 路径表。

[out] Table

指向 MIB_IPPATH_TABLE 结构的指针，该结构包含本地计算机上的 IP 路径条目表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Table</i> 参数中传递 NULL 指针，或者未将 <i>Family</i> 参数指定为 AF_INET、AF_INET6 或 AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	可用内存资源不足，无法完成该操作。
ERROR_NOT_FOUND	未找到 <i>Family</i> 参数中指定的 IP 路径条目。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Family 参数中指定了AF_INET ，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 Family 参数中指定了AF_INET6 ，也会返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

GetIpPathTable 函数在 Windows Vista 及更高版本上定义。

The

GetIpPathTable 函数枚举本地系统上的 IP 路径条目，并在 [MIB_IPPATH_TABLE](#) 结构中返回此信息。

IP 路径条目在 *Table* 参数指向的缓冲区中的[MIB_IPPATH_TABLE](#)结构中返回。

MIB_IPPATH_TABLE 结构包含每个 IP 路径条目的 IP 路径条目计数和[MIB_IPPATH_ROW](#)结构数组。如果不再需要这些返回的结构，请通过调用 [FreeMibTable](#) 释放内存。

Family 参数必须初始化为AF_INET、AF_INET6或AF_UNSPEC。

请注意，*Table* 参数指向的返回[MIB_IPPATH_TABLE](#)结构可能包含 **NumEntries** 成员与 **MIB_IPPATH_TABLE** 结构的Table 成员中第一个[MIB_IPPATH_ROW](#)数组条目之间的对齐填充。**MIB_IPPATH_ROW** 数组条目 之间也可能存在对齐填充。对 **MIB_IPPATH_ROW** 数组条目的任何访问都应假定可能存在填充。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FlushIpPathTable](#)

[FreeMibTable](#)

[GetIpPathEntry](#)

[MIB_IPPATH_ROW](#)

[MIB_IPPATH_TABLE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getIpStatistics 函数 (iphlpapi.h)

项目2023/08/24

GetIpStatistics 函数检索当前计算机的 IP 统计信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetIpStatistics(
    [out] PMIB_IPSTATS Statistics
);
```

parameters

[out] Statistics

指向接收本地计算机的 IP 统计信息 的 MIB_IPSTATS 结构的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	pStats 参数为 NULL，或者 GetIpStatistics 无法写入 pStats 参数指向的内存。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

GetIpStatistics 函数返回当前计算机上 IPv4 的统计信息。在 Windows XP 及更高版本上，GetIpStatisticsEx 可用于获取 IPv4 或 IPv6 的 IP 统计信息。

示例

以下示例检索本地计算机的 IPv4 统计信息，并从返回的数据中输出值。

C++

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIcmpStatistics](#)

[GetIpStatisticsEx](#)

[GetTcpStatistics](#)

[GetUdpStatistics](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPSTATS](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

GetIpStatisticsEx 函数 (iphlpapi.h)

项目2023/08/24

GetIpStatisticsEx 函数检索当前计算机的 Internet 协议 (IP) 统计信息。 GetIpStatisticsEx 函数与 [GetIpStatistics](#) 函数的不同之处在于， GetIpStatisticsEx 还支持 Internet 协议版本 6 (IPv6) 协议系列。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetIpStatisticsEx(
    [out] PMIB_IPSTATS Statistics,
    [in]   ULONG       Family
);
```

parameters

[out] Statistics

指向接收本地计算机的 IP 统计信息 的MIB_IPSTATS 结构的指针。

[in] Family

要检索其统计信息的协议系列。 此参数必须是以下值之一：

值	含义
AF_INET	Internet 协议版本 4 (IPv4)。
AF_INET6	Internet 协议版本 6 (IPv6)。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	pStats 参数为 NULL 或未指向有效内存，或者 dwFamily 参数不是有效值。

ERROR_NOT_SUPPORTED	执行函数调用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetIpStatisticsEx](#) 可用于获取本地计算机上的 IPv4 或 IPv6 的 IP 统计信息。

[GetIpStatistics](#) 函数仅返回本地计算机上 IPv4 的统计信息。

要求

最低受支持的客户端	Windows XP [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2003 [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[GetIpStatistics](#)

[GetTcpStatisticsEx](#)

[GetUdpStatisticsEx](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPSTATS](#)

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获得帮助

getMulticastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

GetMulticastIpAddressEntry 函数检索本地计算机上现有多播 IP 地址条目的信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetMulticastIpAddressEntry(
    [in, out] PMIB_MULTICASTIPADDRESS_ROW Row
);
```

parameters

[in, out] Row

指向多播 IP 地址条目 [MIB_MULTICASTIPADDRESS_ROW](#) 结构条目的指针。成功返回后，将使用现有多播 IP 地址的属性更新此结构。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果 Row 参数指向的 MIB_MULTICASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	参数不正确。如果在 Row 参数中传递 NULL 指针，Row 参数所指向 MIB_MULTICASTIPADDRESS_ROW 的 Address 成员未设置为有效的多播 IPv4 或 IPv6 地址，或者 Row 参数指向的 MIB_MULTICASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员均未指定，则返回此错误。

ERROR_NOT_FOUND	找不到元素。如果 Row 参数指向的 MIB_MULTICASTIPADDRESS_ROW 结构的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口与 MIB_MULTICASTIPADDRESS_ROW 结构中的 Address 成员中指定的 IP 地址和地址系列不匹配，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Row 参数指向的 Address 成员 MIB_MULTICASTIPADDRESS_ROW 指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetMulticastIpAddressEntry](#) 函数在 Windows Vista 及更高版本上定义。

[GetMulticastIpAddressEntry](#) 函数用于检索现有的 [MIB_MULTICASTIPADDRESS_ROW](#) 结构条目。

输入时，Row 参数指向的 [MIB_MULTICASTIPADDRESS_ROW](#) 结构中的 Address 成员必须初始化为有效的多播 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 Row 参数的 [MIB_MULTICASTIPADDRESS_ROW](#) 结构中的至少一个成员：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则使用此成员来确定接口。如果未为 InterfaceLuid 成员设置任何值，(此成员的值) 设置为零，则接下来使用 InterfaceIndex 成员来确定接口。

在调用成功时输出时，[GetMulticastIpAddressEntry](#) 检索多播 IP 地址的其他属性，并填写 Row 参数指向的 [MIB_MULTICASTIPADDRESS_ROW](#) 结构。

可以调用 [GetMulticastIpAddressTable](#) 函数来枚举本地计算机上的多播 IP 地址条目。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows

标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[GetMulticastIpAddressTable](#)

[MIB_MULTICASTIPADDRESS_ROW](#)

[MIB_MULTICASTIPADDRESS_TABLE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

GetMulticastIpAddressTable 函数 (netioapi.h)

项目2023/08/25

GetMulticastIpAddressTable 函数检索本地计算机上的多播 IP 地址表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetMulticastIpAddressTable(
    [in] ADDRESS_FAMILY Family,
    [out] PMIB_MULTICASTIPADDRESS_TABLE *Table
);
```

parameters

[in] Family

要检索的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和PF_ 协议系列常量的值 (相同，例如，AF_INET 和 PF_INET)，因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 Ws2def.h 头文件中定义了此成员的可能值。请注意，Ws2def.h 头文件会自动包含在 Winsock2.h 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	地址系列未指定。指定此参数时，此函数返回包含 IPv4 和 IPv6 条目的多播 IP 地址表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数时，此函数返回仅包含 IPv4 条目的多播 IP 地址表。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数时，此函数返回仅包含 IPv6 条目的多播 IP 地址表。

[out] Table

指向 [MIB_MULTICASTIPADDRESS_TABLE](#) 结构的指针，该结构包含本地计算机上的任意广播 IP 地址条目的表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Table</i> 参数中传递 NULL 指针，或者 <i>Family</i> 参数未指定为AF_INET、AF_INET6或AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存资源不足，无法完成操作。
ERROR_NOT_FOUND	未找到 <i>Family</i> 参数中指定的任何广播 IP 地址条目。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且已在 <i>Family</i> 参数中指定了AF_INET，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且已在 <i>Family</i> 参数中指定了AF_INET6，则也会返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`GetMulticastIpAddressTable` 函数是在 Windows Vista 及更高版本上定义的。

The

`GetMulticastIpAddressTable` 函数枚举本地系统上的多播 IP 地址，并在 [MIB_MULTICASTIPADDRESS_TABLE](#) 结构中返回此信息。

多播 IP 地址条目在 *Table* 参数指向的缓冲区中的[MIB_MULTICASTIPADDRESS_TABLE](#)结构中返回。[MIB_MULTICASTIPADDRESS_TABLE](#) 结构包含多播 IP 地址条目计数和每个多播 IP 地址条目的[MIB_MULTICASTIPADDRESS_ROW](#)结构数组。当不再需要这些返回的结构时，通过调用 [FreeMibTable](#) 释放内存。

Family 参数必须初始化为 AF_INET、AF_INET6 或 AF_UNSPEC。

请注意，*Table* 参数指向的返回 MIB_MULTICASTIPADDRESS_TABLE 结构可能包含 NumEntries 成员与 MIB_MULTICASTIPADDRESS_TABLE 结构的 Table 成员中的第一个 MIB_MULTICASTIPADDRESS_ROW 数组条目之间的对齐填充。MIB_MULTICASTIPADDRESS_ROW 数组条目之间也可能存在对齐的填充。对 MIB_MULTICASTIPADDRESS_ROW 数组条目的任何访问都应假定可能存在填充。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[FreeMibTable](#)

[GetMulticastIpAddressEntry](#)

[MIB_MULTICASTIPADDRESS_ROW](#)

[MIB_MULTICASTIPADDRESS_TABLE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

GetNetworkConnectivityHint 函数 (netioapi.h)

项目2023/08/25

检索应用程序或服务可能遇到的网络连接的聚合级别和成本。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetNetworkConnectivityHint(
    [out] NL_NETWORK_CONNECTIVITY_HINT *ConnectivityHint
);
```

parameters

[out] ConnectivityHint

指向 [NL_NETWORK_CONNECTIVITY_HINT](#) 类型的值的指针。 函数将此值设置为聚合连接级别和成本提示。

返回值

在用户模式下，成功时返回 `NO_ERROR`，失败时返回 Win32 错误代码。 在内核模式下，成功时返回 `STATUS_SUCCESS`，失败时返回 NTSTATUS 错误代码。

要求

最低受支持的客户端	Windows 10 版本 2004 (10.0; 内部版本 19041)
最低受支持的服务器	Windows Server 版本 2004 (10.0; 内部版本 19041)
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib

DLL

lphlpapi.dll

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

GetNetworkConnectivityHintForInterface 函数 (netioapi.h)

项目2023/08/25

检索指定接口的网络连接级别和成本。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API  
GetNetworkConnectivityHintForInterface(  
    [in]  NET_IFINDEX           InterfaceIndex,  
    [out] NL_NETWORK_CONNECTIVITY_HINT *ConnectivityHint  
) ;
```

parameters

[in] InterfaceIndex

类型的值 `NET_IFINDEX` 表示要检索其连接信息的接口的索引。

[out] ConnectivityHint

指向 `NL_NETWORK_CONNECTIVITY_HINT` 类型的值的指针。 函数将此值设置为指定接口的连接级别和成本提示。

返回值

在用户模式下， 在成功时返回 `NO_ERROR`， 在失败时返回 Win32 错误代码。 在内核模式下， 在成功时返回 `STATUS_SUCCESS`， 在失败时返回 `NTSTATUS` 错误代码。

要求

最低受支持的客户端	Windows 10, 版本 2004 (10.0;内部版本 19041)
最低受支持的服务器	Windows Server 版本 2004 (10.0;内部版本 19041)

目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getNetworkParams 函数 (iphlpapi.h)

项目2023/08/24

GetNetworkParams 函数检索本地计算机的网络参数。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetNetworkParams(
    [out] PFIXED_INFO pFixedInfo,
    [in]   PULONG     pOutBufLen
);
```

parameters

[out] pFixedInfo

指向包含 [FIXED_INFO](#) 结构的缓冲区的指针，该结构接收本地计算机的网络参数（如果函数成功）。调用 [GetNetworkParams](#) 函数之前，调用方必须分配此缓冲区。

[in] pOutBufLen

指向 [ULONG](#) 变量的指针，该变量指定 [FIXED_INFO](#) 结构的大小。如果此大小不足以保存信息，则 [GetNetworkParams](#) 会用所需的大小填充此变量，并返回 [ERROR_BUFFER_OVERFLOW](#) 错误代码。

返回值

如果函数成功，则返回值 [ERROR_SUCCESS](#)。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_BUFFER_OVERFLOW	用于接收网络参数信息的缓冲区太小。如果 <i>pOutBufLen</i> 参数太小而无法保存网络参数信息或 <i>pFixedInfo</i> 参数为 NULL 指针，则返回此值。返回此错误代码时， <i>pOutBufLen</i> 参数指向所需的缓冲区大小。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>pOutBufLen</i> 参数为 NULL 指针，调用进程对 <i>pOutBufLen</i> 所指向的内存没有

	读/写访问权限，或者调用进程对 <i>pFixedInfo</i> 参数所指向的内存没有写入访问权限，则返回此错误。
ERROR_NO_DATA	本地计算机不存在网络参数信息。
ERROR_NOT_SUPPORTED	在本地计算机上运行的操作系统不支持 <i>GetNetworkParams</i> 函数。
其他	如果函数失败，请使用 FormatMessage 获取返回错误的消息字符串。

注解

GetNetworkParams 函数用于检索本地计算机的网络参数。 网络参数以 [FIXED_INFO](#) 结构返回。 [FIXED_INFO](#) 结构的内存必须由应用程序分配。 应用程序负责在不再需要内存时释放此内存。

在 Microsoft Windows 软件开发工具包 (SDK) 中，定义了 [FIXED_INFO_WIN2KSP1](#) 结构。 如果目标平台是 Windows 2000 且 Service Pack 1 (SP1) 及更高版本 (`NTDDI_VERSION >= NTDDI_WIN2KSP1`、`_WIN32_WINNT >= 0x0501` 或 `WINVER >= 0x0501`)，则编译应用程序时，[FIXED_INFO_WIN2KSP1](#) 结构的类型为 [FIXED_INFO](#) 结构。 如果目标平台不是 Windows 2000 SP1 及更高版本，则编译应用程序时，[FIXED_INFO](#) 结构未定义。

Windows 98 及更高版本支持 *GetNetworkParams* 函数和 [FIXED_INFO](#) 结构。 但是，若要为早于 Windows 2000 且 Service Pack 1 (SP1) 的目标平台生成应用程序，必须使用早期版本的平台软件开发工具包 (SDK)。

示例

以下示例检索本地计算机的网络参数，并从返回的数据中输出信息。

C++

```

// 
// Link with IPHlpAPI.lib
//
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <windows.h>
#pragma comment(lib, "IPHLPAPI.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

```

```
int __cdecl main()
{
    FIXED_INFO *pFixedInfo;
    ULONG ulOutBufLen;
    DWORD dwRetVal;
    IP_ADDR_STRING *pIPAddr;

    pFixedInfo = (FIXED_INFO *) MALLOC(sizeof(FIXED_INFO));
    if (pFixedInfo == NULL) {
        printf("Error allocating memory needed to call GetNetworkParams\n");
        return 1;
    }
    ulOutBufLen = sizeof(FIXED_INFO);

    // Make an initial call to GetAdaptersInfo to get
    // the necessary size into the ulOutBufLen variable
    if (GetNetworkParams(pFixedInfo, &ulOutBufLen) == ERROR_BUFFER_OVERFLOW)
    {
        FREE(pFixedInfo);
        pFixedInfo = (FIXED_INFO *) MALLOC(ulOutBufLen);
        if (pFixedInfo == NULL) {
            printf("Error allocating memory needed to call
GetNetworkParams\n");
            return 1;
        }
    }

    if (dwRetVal = GetNetworkParams(pFixedInfo, &ulOutBufLen) == NO_ERROR) {

        printf("Host Name: %s\n", pFixedInfo->HostName);
        printf("Domain Name: %s\n", pFixedInfo->DomainName);

        printf("DNS Servers:\n");
        printf("\t%s\n", pFixedInfo->DnsServerListIpAddress.String);

        pIPAddr = pFixedInfo->DnsServerList.Next;
        while (pIPAddr) {
            printf("\t%s\n", pIPAddr->IpAddress.String);
            pIPAddr = pIPAddr->Next;
        }

        printf("Node Type: ");
        switch (pFixedInfo->NodeType) {
        case BROADCAST_NODETYPE:
            printf("Broadcast node\n");
            break;
        case PEER_TO_PEER_NODETYPE:
            printf("Peer to Peer node\n");
            break;
        case MIXED_NODETYPE:
            printf("Mixed node\n");
            break;
        case HYBRID_NODETYPE:
            printf("Hybrid node\n");
            break;
        }
    }
}
```

```

        break;
    default:
        printf("Unknown node type %0lx\n", pFixedInfo->NodeType);
        break;
    }

    printf("DHCP scope name: %s\n", pFixedInfo->ScopeId);

    if (pFixedInfo->EnableRouting)
        printf("Routing: enabled\n");
    else
        printf("Routing: disabled\n");

    if (pFixedInfo->EnableProxy)
        printf("ARP proxy: enabled\n");
    else
        printf("ARP Proxy: disabled\n");

    if (pFixedInfo->EnableDns)
        printf("DNS: enabled\n");
    else
        printf("DNS: disabled\n");

} else {
    printf("GetNetworkParams failed with error: %d\n", dwRetVal);
    return 1;
}

if (pFixedInfo)
    FREE(pFixedInfo);

return 0;
}

```

要求

最低受支持的客户端	Windows 2000 专业版 [桌面应用 UWP 应用]
最低受支持的服务器	Windows 2000 Server [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[FIXED_INFO](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

反馈

此页面是否有帮助?

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

getNumberOfInterfaces 函数 (iphlpapi.h)

项目2023/08/24

GetNumberOfInterfaces 函数检索本地计算机上的接口数。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetNumberOfInterfaces(
    [out] PDWORD pdwNumIf
);
```

parameters

[out] pdwNumIf

指向接收本地计算机上接口数的 DWORD 变量的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，请使用 FormatMessage 获取返回错误的消息字符串。

注解

GetNumberOfInterfaces 函数返回本地计算机上的接口数，包括环回接口。此数字比 GetAdaptersInfo 和 GetInterfaceInfo 函数返回的适配器数多一个，因为这些函数不返回有关环回接口的信息。

要求

最低受支持的客户端

Windows 2000 Professional [仅限桌面应用]

最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetAdaptersInfo](#)

[GetIfEntry](#)

[GetInterfaceInfo](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

GetOwnerModuleFromTcpEntry 函数 (iphlpapi.h)

项目2023/08/24

GetOwnerModuleFromTcpEntry 函数检索有关在 MIB 表行中为特定 IPv4 TCP 终结点发出上下文绑定的模块的数据。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetOwnerModuleFromTcpEntry(
    [in]      PMIB_TCPROW_OWNER_MODULE     pTcpEntry,
    [in]      TCPIP_OWNER_MODULE_INFO_CLASS Class,
    [out]     PVOID                      pBuffer,
    [in, out] PDWORD                    pdwSize
);
```

parameters

[in] pTcpEntry

指向 [MIB_TCPROW_OWNER_MODULE](#) 结构的指针，该结构包含用于获取所有者模块的 IPv4 TCP 终结点条目。

[in] Class

一个[TCPIP_OWNER_MODULE_INFO_CLASS](#)枚举值，该值指示要获取的有关所有者模块的数据的类型。 [TCPIP_OWNER_MODULE_INFO_CLASS](#)枚举在 *lprtrmib.h* 头文件中定义。

此参数必须设置为 [TCPIP_OWNER_MODULE_INFO_BASIC](#)。

[out] pBuffer

一个指针，一个缓冲区，其中包含具有所有者模块数据的 [TCPIP_OWNER_MODULE_BASIC_INFO](#) 结构。在此缓冲区中返回的数据类型由 *Class* 参数的值指示。

将 *Class* 设置为相应的值时，以下结构用于 *Buffer* 中的数据。

类枚举值

缓冲区数据格式

[in, out] pdwSize

缓冲区中返回的结构的估计大小（以字节为单位）。如果此值设置得太小，则此函数将返回ERROR_INSUFFICIENT_BUFFER，并且此字段将包含正确的缓冲区大小。所需的大小是相应结构的大小加上与结构中指向的数据长度相等的额外字节数（例如，名称和路径字符串）。

返回值

如果函数调用成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	为表分配的空间不足。表的大小在pdwSize参数中返回，并且必须在后续调用此函数时使用，才能成功检索表。
ERROR_INVALID_PARAMETER	参数不正确。如果pTcpEntry或pdwSize参数之一为NULL，则返回此值。如果Class参数不等于TCPIP_OWNER_MODULE_INFO_BASIC，也会返回此值。
ERROR_NOT_ENOUGH_MEMORY	没有足够的可用内存来完成该操作。
ERROR_NOT_FOUND	找不到元素。如果pTcpEntry参数指向的MIB_TCPOROWNER_MODULE结构的dwOwningPid成员为零或找不到，则返回此值。
ERROR_PARTIAL_COPY	只完成了请求的一部分。

注解

Buffer参数不仅包含一个结构，其中包含指向特定数据的指针，例如，指向包含所有者模块名称和路径的零终止字符串的指针，还包含实际数据本身；即名称和路径字符串。因此，在计算缓冲区大小时，请确保为结构以及结构成员指向的数据提供足够的空间。

最佳做法是将TCP表条目解析为所有者模块。在少数情况下，在TCPIP_OWNER_MODULE_BASIC_INFO结构中返回的所有者模块名称可以是进程名称（如“svchost.exe”、服务名称（如“RPC”）或组件名称（如“timer.dll”）。

对于在Windows Vista或更高版本上运行的计算机，GetOwnerModuleFromTcpEntry函数检索的TCPIP_OWNER_MODULE_BASIC_INFO的pModuleName和pModulePath成员

可能指向某些 TCP 连接的空字符串。默认情况下，启动位于 Windows 系统文件夹 (C:\Windows\System32 中的 TCP 连接的应用程序被视为受保护。如果 GetOwnerModuleFromTcpEntry 函数由不是 Administrators 组成员的用户调用，则函数调用将成功，但 pModuleName 和 pModulePath 成员将指向包含受保护应用程序启动的 TCP 连接的空字符串的内存。

对于在 Windows Vista 或更高版本上运行的计算机，访问 [TCPIP_OWNER_MODULE_BASIC_INFO](#) 结构的 pModuleName 和 pModulePath 成员受用户帐户控制 (UAC) 的限制。如果调用此函数的应用程序由作为管理员组成员（而非内置管理员）登录的用户执行，则此调用将成功，但对这些成员的访问将返回空字符串，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果 Windows Vista 或更高版本上的应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能访问受保护的 pModuleName 和 pModulePath 成员。

要求

最低受支持的客户端	Windows Vista、Windows XP 和 SP2 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[MIB_TCPROW_OWNER_MODULE](#)

[TCPIP_OWNER_MODULE_BASIC_INFO](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

GetOwnerModuleFromTcp6Entry 函数 (iphlpapi.h)

项目2023/08/24

GetOwnerModuleFromTcp6Entry 函数检索有关在 MIB 表行中为特定 IPv6 TCP 终结点发出上下文绑定的模块的数据。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetOwnerModuleFromTcp6Entry(
    [in]      PMIB_TCP6ROW_OWNER_MODULE     pTcpEntry,
    [in]      TCPIP_OWNER_MODULE_INFO_CLASS Class,
    [out]     PVOID                      pBuffer,
    [in, out] PDWORD                    pdwSize
);
```

parameters

[in] pTcpEntry

指向 [MIB_TCP6ROW_OWNER_MODULE](#) 结构的指针，该结构包含用于获取所有者模块的 IPv6 TCP 终结点条目。

[in] Class

一个 [TCPIP_OWNER_MODULE_INFO_CLASS](#) 枚举值，该值指示要获取的有关所有者模块的数据的类型。 [TCPIP_OWNER_MODULE_INFO_CLASS](#) 枚举在 *lprtrmib.h* 头文件中定义。

此参数必须设置为 [TCPIP_OWNER_MODULE_INFO_BASIC](#)。

[out] pBuffer

指向缓冲区的指针，该缓冲区包含具有所有者模块数据的 [TCPIP_OWNER_MODULE_BASIC_INFO](#) 结构。在此缓冲区中返回的数据类型由 *Class* 参数的值指示。

将 *Class* 设置为相应的值时，以下结构用于 *Buffer* 中的数据。

类枚举值

缓冲区数据格式

[in, out] pdwSize

缓冲区中返回的结构的估计大小（以字节为单位）。如果此值设置得太小，则此函数返回ERROR_INSUFFICIENT_BUFFER，并且此字段将包含正确的结构大小。

返回值

如果函数调用成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	为表分配的空间不足。表的大小在pdwSize参数中返回，并且必须在后续调用此函数时使用，才能成功检索表。
ERROR_INVALID_PARAMETER	参数不正确。如果pTcpEntry或pdwSize参数之一为NULL，则返回此值。如果Class参数不等于TCPIP_OWNER_MODULE_INFO_BASIC，也会返回此值。
ERROR_NOT_ENOUGH_MEMORY	没有足够的可用内存来完成该操作。
ERROR_NOT_FOUND	找不到该元素。如果pTcpEntry参数指向的MIB_TCP6ROW_OWNER_MODULE的dwOwningPid成员为零或找不到，则返回此值。
ERROR_PARTIAL_COPY	只完成了请求的一部分。

注解

Buffer参数不仅包含一个结构，其中包含指向特定数据的指针，例如，指向包含所有者模块名称和路径的零终止字符串的指针，还包含实际数据本身；即名称和路径字符串。因此，在计算缓冲区的大小时，请确保有足够的空间来容纳结构以及结构成员指向的数据。

最佳做法是将TCP表条目解析为所有者模块。在少数情况下，TCPIP_OWNER_MODULE_BASIC_INFO结构中返回的所有者模块名称可以是进程名称（“svchost.exe”）、服务名称（如“RPC”）或组件名称（如“timer.dll”）。

对于在Windows Vista或更高版本上运行的计算机，GetOwnerModuleFromTcpEntry函数检索的TCPIP_OWNER_MODULE_BASIC_INFO的pModuleName和pModulePath成员可能指向某些TCP连接的空字符串。默认情况下，启动位于Windows系统文件夹(C:\Windows\System32)中的TCP连接的应用程序被视为受保护。如果

`GetOwnerModuleFromTcpEntry` 函数由不是 Administrators 组成员的用户调用，则函数调用将成功，但 `pModuleName` 和 `pModulePath` 成员将指向包含受保护应用程序启动的 TCP 连接的空字符串的内存。

对于在 Windows Vista 或更高版本上运行的计算机，访问 `TCPIP_OWNER_MODULE_BASIC_INFO` 结构的 `pModuleName` 和 `pModulePath` 成员受用户帐户控制 (UAC) 的限制。如果调用此函数的应用程序由作为管理员组成员（而非内置管理员）登录的用户执行，则此调用将成功，但对这些成员的访问将返回空字符串，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果 Windows Vista 或更高版本上的应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能访问受保护的 `pModuleName` 和 `pModulePath` 成员。

要求

最低受支持的客户端	Windows Vista、Windows XP 和 SP2 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[MIB_TCP6ROW_OWNER_MODULE](#)

[TCPIP_OWNER_MODULE_INFO_CLASS](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

GetOwnerModuleFromUdpEntry 函数 (iphlpapi.h)

项目2023/08/24

GetOwnerModuleFromUdpEntry 函数检索有关在 MIB 表行中为特定 IPv4 UDP 终结点发出上下文绑定的模块的数据。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetOwnerModuleFromUdpEntry(
    [in]      PMIB_UDPROW_OWNER_MODULE     pUdpEntry,
    [in]      TCPIP_OWNER_MODULE_INFO_CLASS Class,
    [out]     PVOID                      pBuffer,
    [in, out] PDWORD                    pdwSize
);
```

parameters

[in] pUdpEntry

指向 [MIB_UDPROW_OWNER_MODULE](#) 结构的指针，该结构包含用于获取所有者模块的 IPv4 UDP 终结点条目。

[in] Class

一个[TCPIP_OWNER_MODULE_INFO_CLASS](#)枚举值，该值指示要获取的有关所有者模块的数据的类型。

[out] pBuffer

包含具有所有者模块数据的 [TCPIP_OWNER_MODULE_BASIC_INFO](#) 结构的缓冲区。在此缓冲区中返回的数据类型由 *Class* 参数的值指示。

将 *Class* 设置为相应的值时，以下结构用于 *Buffer* 中的数据。

类枚举值	缓冲区数据格式
TCPIP_OWNER_MODULE_BASIC_INFO	TCPIP_OWNER_MODULE_BASIC_INFO

[in, out] pdwSize

缓冲区中返回的结构的估计大小（以字节为单位）。如果此值设置得太小，则此函数返回 **ERROR_INSUFFICIENT_BUFFER**，并且此字段将包含正确的结构大小。

返回值

如果调用成功，则返回 **NO_ERROR** 的值。否则，将返回以下错误。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	为表分配的空间不足。表的大小在 <i>pdwSize</i> 参数中返回，并且必须在后续调用此函数时使用，才能成功检索表。

注解

Buffer 参数不仅包含具有指向特定数据的指针的结构，例如，指向包含所有者模块名称和路径的零终止字符串的指针，还包含实际数据本身；即名称和路径字符串。因此，在计算缓冲区大小时，请确保为结构以及结构成员指向的数据提供足够的空间。

最佳做法是将 UDP 表条目解析为所有者模块。在少数情况下，[TCPIP_OWNER_MODULE_BASIC_INFO](#) 结构中返回的所有者模块名称可以是进程名称（如“svchost.exe”、服务名称（如“RPC”）或组件名称（例如“timer.dll”）。

对于在 Windows Vista 或更高版本上运行的计算机，访问

[TCPIP_OWNER_MODULE_BASIC_INFO](#) 结构的 **pModuleName** 和 **pModulePath** 成员受用户帐户控制 (UAC) 的限制。如果调用此函数的应用程序由作为管理员组成员（而非内置管理员）登录的用户执行，则此调用将成功，但对这些成员的访问将返回空字符串，除非应用程序已在清单文件中标记为 **requestedExecutionLevel** 设置为 **requireAdministrator**。如果 Windows Vista 或更高版本上的应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能访问受保护的 **pModuleName** 和 **pModulePath** 成员。

要求

最低受支持的客户端	Windows Vista、Windows XP 和 SP2 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]
目标平台	Windows

标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

反馈

此页面是否有帮助?



[在 Microsoft Q&A 获得帮助](#)

GetOwnerModuleFromUdp6Entry 函数 (iphlpapi.h)

项目2023/08/24

GetOwnerModuleFromUdp6Entry 函数检索有关在 MIB 表行中为特定 IPv6 UDP 终结点发出上下文绑定的模块的数据。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetOwnerModuleFromUdp6Entry(
    [in]      PMIB_UDP6ROW_OWNER_MODULE     pUdpEntry,
    [in]      TCPIP_OWNER_MODULE_INFO_CLASS Class,
    [out]     PVOID                      pBuffer,
    [in, out] PDWORD                    pdwSize
);
```

parameters

[in] pUdpEntry

指向 [MIB_UDP6ROW_OWNER_MODULE](#) 结构的指针，该结构包含用于获取所有者模块的 IPv6 UDP 终结点条目。

[in] Class

[TCPIP_OWNER_MODULE_INFO_CLASS](#) 枚举值，该值指示要获取的有关所有者模块的数据的类型。

[out] pBuffer

包含具有所有者模块数据的 [TCPIP_OWNER_MODULE_BASIC_INFO](#) 结构的缓冲区。在此缓冲区中返回的数据类型由 *Class* 参数的值指示。

将 *Class* 设置为相应的值时，以下结构用于 *Buffer* 中的数据。

类枚举值	缓冲区数据格式
TCPIP_OWNER_MODULE_BASIC_INFO	TCPIP_OWNER_MODULE_BASIC_INFO

[in, out] pdwSize

缓冲区中返回的结构的估计大小（以字节为单位）。如果此值设置得太小，则此函数将返回 `ERROR_INSUFFICIENT_BUFFER`，并且此字段将包含结构的正确大小。

返回值

如果调用成功，则返回 `NO_ERROR` 的值。否则，将返回以下错误。

返回代码	说明
<code>ERROR_INSUFFICIENT_BUFFER</code>	为表分配的空间不足。表的大小在 <code>pdwSize</code> 参数中返回，并且必须在后续调用此函数时使用，才能成功检索表。

注解

`Buffer` 参数不仅包含一个结构，其中包含指向特定数据的指针，例如，指向包含所有者模块名称和路径的零终止字符串的指针，还包含实际数据本身；即名称和路径字符串。因此，在计算缓冲区大小时，请确保为结构以及结构成员指向的数据提供足够的空间。

最佳做法是将 UDP 表条目解析为所有者模块。在少数情况下，`TCPIP_OWNER_MODULE_BASIC_INFO` 结构中返回的所有者模块名称可以是进程名称（如“svchost.exe”、服务名称（如“RPC”）或组件名称（例如“timer.dll”）。

对于在 Windows Vista 或更高版本上运行的计算机，访问

`TCPIP_OWNER_MODULE_BASIC_INFO` 结构的 `pModuleName` 和 `pModulePath` 成员受用户帐户控制 (UAC) 的限制。如果调用此函数的应用程序由作为管理员组成员（而非内置管理员）登录的用户执行，则此调用将成功，但对这些成员的访问将返回空字符串，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果 Windows Vista 或更高版本上的应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能访问受保护的 `pModuleName` 和 `pModulePath` 成员。

要求

最低受支持的客户端	Windows Vista、Windows XP 和 SP2 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]
目标平台	Windows

标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

反馈

此页面是否有帮助?



[在 Microsoft Q&A 获得帮助](#)

getPerAdapterInfo 函数 (iphlpapi.h)

项目2023/08/24

GetPerAdapterInfo 函数检索与指定接口对应的适配器的相关信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetPerAdapterInfo(
    [in]    ULONG             IfIndex,
    [out]   PIP_PER_ADAPTER_INFO pPerAdapterInfo,
    [in]    PULONG            pOutBufLen
);
```

parameters

[in] IfIndex

接口的索引。 GetPerAdapterInfo 函数检索与此接口对应的适配器的信息。

[out] pPerAdapterInfo

指向接收适配器相关信息 的 IP_PER_ADAPTER_INFO 结构的指针。

[in] pOutBufLen

指向 ULONG 变量的指针，该变量指定 IP_PER_ADAPTER_INFO 结构的大小。如果此大小不足以保存信息，则 GetPerAdapterInfo 会用所需的大小填充此变量，并返回 ERROR_BUFFER_OVERFLOW 错误代码。

返回值

如果函数成功，则返回值为 ERROR_SUCCESS。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_BUFFER_OVERFLOW	pOutBufLen 参数指示的缓冲区大小太小，无法容纳适配器信息。 pOutBufLen 参数指向所需的大小。

ERROR_INVALID_PARAMETER	<i>pOutBufLen</i> 参数为 NULL , 或者调用进程对 <i>pOutBufLen</i> 指向的内存没有读/写访问权限, 或者调用进程对 <i>pAdapterInfo</i> 参数指向的内存没有写入访问权限。
ERROR_NOT_SUPPORTED	在本地计算机上运行的操作系统不支持 GetPerAdapterInfo 。
其他	如果函数失败, 请使用 FormatMessage 获取返回错误的消息字符串。

注解

适配器索引可能会在禁用然后启用适配器时发生更改, 或者在其他情况下, 不应被视为永久性。

要求

最低受支持的客户端	Windows 2000 专业版 [桌面应用 UWP 应用]
最低受支持的服务器	Windows 2000 Server [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IP_PER_ADAPTER_INFO](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

GetPerTcp6ConnectionEStats 函数 (iphlpapi.h)

项目2023/08/24

GetPerTcp6ConnectionEStats 函数检索 IPv6 TCP 连接的扩展统计信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetPerTcp6ConnectionEStats(
    PMIB_TCP6ROW      Row,
    TCP_ESTATS_TYPE   EstatsType,
    [out] PUCHAR       Rw,
    ULONG             RwVersion,
    ULONG             RwSize,
    [out] PUCHAR       Ros,
    ULONG             RosVersion,
    ULONG             RosSize,
    [out] PUCHAR       Rod,
    ULONG             RodVersion,
    ULONG             RodSize
);
```

parameters

Row

指向 IPv6 TCP 连接的 [MIB_TCP6ROW](#) 结构的指针。

EstatsType

请求的 TCP 扩展统计信息的类型。如果调用成功，此参数确定在 *Rw*、*Rod* 和 *Ros* 参数中返回的信息的数据和格式。

此参数可以是 *Tcpestats.h* 头文件中定义的 [TCP_ESTATS_TYPE](#) 枚举类型的值之一。

值	含义
TcpConnectionEStatsSynOpts	此值请求 TCP 连接的 SYN 交换信息。 此枚举值仅提供只读静态信息。

	<p>如果 <i>Ros</i> 参数不为 NULL 且函数成功, 则 <i>Ros</i> 参数指向的缓冲区应包含 TCP_ESTATS_SYN_OPTS_ROS_v0 结构。</p>
TcpConnectionEstatsData	<p>此值请求 TCP 连接的扩展数据传输信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_DATA_RW_v0 结构。</p> <p>如果为此 TCP 连接启用了扩展数据传输信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_DATA_ROD_v0 结构。</p>
TcpConnectionEstatsSndCong	<p>此值请求 TCP 连接的发送方拥塞。 (只读静态、只读动态和读/写信息的所有三种类型的信息都可用于此枚举值。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS SND_CONG_RW_v0 结构。</p> <p>如果 <i>Ros</i> 参数不为 NULL 且函数成功, 则 <i>Ros</i> 参数指向的缓冲区应包含 TCP_ESTATS SND_CONG_ROS_v0 结构。</p> <p>如果为此 TCP 连接启用了发送方拥塞信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS SND_CONG_ROD_v0 结构。</p>
TcpConnectionEstatsPath	<p>此值请求 TCP 连接的扩展路径度量信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_PATH_RW_v0 结构。</p> <p>如果为此 TCP 连接启用了扩展路径度量信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_PATH_ROD_v0 结构。</p>
TcpConnectionEstatsSendBuff	<p>此值请求 TCP 连接的扩展输出队列信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_SEND_BUFF_RW_v0 结构。</p> <p>如果为此 TCP 连接启用了扩展输出队列信息, 则 <i>Rod</i> 参数不为 NULL, 并且函数成功, <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_SEND_BUFF_ROD_v0 结构。</p>
TcpConnectionEstatsRec	<p>此值请求 TCP 连接的扩展本地接收器信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_REC_RW_v0 结构。</p>

	<p>如果为此 TCP 连接启用了扩展的本地接收器信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, 则 <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_REC_ROD_v0 结构。</p>
TcpConnectionEstatsObsRec	<p>此值请求 TCP 连接的扩展远程接收方信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_OBS_REC_RW_v0 结构。</p>
TcpConnectionEstatsBandwidth	<p>此值请求带宽上的 TCP 连接的带宽估计统计信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_BANDWIDTH_RW_v0 结构。</p>
TcpConnectionEstatsFineRtt	<p>此值请求 TCP 连接的精细往返时间 (RTT) 估计统计信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_FINE_RTT_RW_v0 结构。</p> <p>如果为此 TCP 连接启用了细粒度 RTT 估计统计信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, 则 <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_FINE_RTT_ROD_v0 结构。</p>

[out] *Rw*

指向用于接收读/写信息的缓冲区的指针。如果应用程序不想检索 TCP 连接的读/写信息, 此参数可能是 **NULL** 指针。

RwVersion

请求的读/写信息的版本。当前支持的值为 0 的版本。

RwSize

Rw 参数指向的缓冲区的大小 (以字节为单位)。

[out] *Ros*

指向用于接收只读静态信息的缓冲区的指针。如果应用程序不想检索 TCP 连接的只读静态信息，此参数可能是 **NULL** 指针。

RosVersion

请求的只读静态信息的版本。当前支持的值为 0 的版本。

RosSize

Ros 参数指向的缓冲区的大小（以字节为单位）。

[out] Rod

指向用于接收只读动态信息的缓冲区的指针。如果应用程序不想检索 TCP 连接的只读动态信息，此参数可能是 **NULL** 指针。

RodVersion

请求的只读动态信息的版本。当前支持的值为 0 的版本。

RodSize

Rod 参数指向的缓冲区的大小（以字节为单位）。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	传递给函数的缓冲区太小。如果 <i>Rw</i> 、 <i>Ros</i> 或 <i>Rod</i> 参数指向的缓冲区不够大，无法接收数据，则返回此错误。如果 <i>Rw</i> 、 <i>Ros</i> 或 <i>Rod</i> 参数指向的给定缓冲区之一为 NULL ，但在关联的 <i>RwSize</i> 、 <i>RosSize</i> 或 <i>RodSize</i> 中指定了长度，则也会返回此错误。 此错误值在 Windows Vista 和 Windows Server 2008 上返回。
ERROR_INVALID_PARAMETER	参数不正确。如果 <i>Row</i> 参数为 NULL 指针，则返回此错误。
ERROR_INVALID_USER_BUFFER	提供给请求操作的用户缓冲区无效。如果 <i>Rw</i> 、 <i>Ros</i> 或 <i>Rod</i> 参数指向的给定缓冲区之一为 NULL ，但在关联的 <i>RwSize</i> 、 <i>RosSize</i> 或 <i>RodSize</i> 中指定了长度，则返回此错误。因此，如果满足以下任一条件，则返回此错误：

- *Row* 参数为 NULL 指针, *RwSize* 参数为非零值。
- *Ros* 参数为 NULL 指针, *RosSize* 参数为非零值。
- *Rod* 参数为 NULL 指针, *RodSize* 参数为非零值。

此错误值在 Windows 7 和 Windows Server 2008 R2 上返回。

ERROR_NOT_FOUND	找不到此请求的条目。如果找不到 <i>Row</i> 参数中指定的 TCP 连接，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果 <i>RwVersion</i> 、 <i>RosVersion</i> 或 <i>RodVersion</i> 参数未设置为零，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetPerTcp6ConnectionEStats](#) 函数在 Windows Vista 及更高版本上定义。

[GetPerTcp6ConnectionEStats](#) 函数旨在使用 TCP 诊断网络和应用程序中的性能问题。如果基于网络的应用程序性能不佳，TCP 可以确定瓶颈是在发送方、接收方还是网络本身。如果瓶颈在网络中，TCP 可以提供有关其性质的特定信息。

[GetPerTcp6ConnectionEStats](#) 函数检索 *Row* 参数中传递的 IPv6 TCP 连接的扩展统计信息。检索的扩展统计信息的类型在 *EstatsType* 参数中指定。以前必须通过调用 [SetPerTcp6ConnectionEStats](#) 函数为所有 [TCP_ESTATS_TYPE](#) 值启用此 TCP 连接的扩展统计信息，除非在 *EstatsType* 参数中传递 [TcpConnectionEstatsSynOpts](#)。

[GetTcp6Table](#) 函数用于检索本地计算机上的 IPv6 TCP 连接表。此函数返回包含 [MIB_TCP6ROW](#) 项数组的 [MIB_TCP6TABLE](#) 结构。传递给 [GetPerTcp6ConnectionEStats](#) 函数的 *Row* 参数必须是现有 IPv6 TCP 连接的条目。

当前支持的唯一 TCP 连接统计信息版本为版本零。因此，传递给 [GetPerTcp6ConnectionEStats](#) 的 *RwVersion*、*RosVersion* 和 *RodVersion* 参数应设置为 0。

有关 IPv4 连接上的扩展 TCP 统计信息的信息，请参阅 [GetPerTcpConnectionEStats](#) 和 [SetPerTcpConnectionEStats](#) 函数。

[SetPerTcp6ConnectionEStats](#) 函数只能由以 Administrators 组成员身份登录的用户调用。如果 [SetPerTcp6ConnectionEStats](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 [ERROR_ACCESS_DENIED](#)。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 [requestedExecutionLevel](#) 设置为 [requireAdministrator](#)。如果应用程序缺少此清

单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员（RunAs 管理员）此函数才能成功。

`GetPerTcp6ConnectionEStats` 的调用方应在返回的 `Rw` 结构中检查 `EnableCollection` 字段，如果不是 `TRUE`，则调用方应忽略 `Ros` 和 `Rod` 结构中的数据。如果 `EnableCollection` 设置为 `FALSE`，则 `Ros` 和 `Rod` 中返回的数据未定义。例如，发生此情况的一个条件是，使用 `GetPerTcp6ConnectionEStats` 检索 IPv6 TCP 连接的扩展统计信息，并且之前调用 `SetPerTcp6ConnectionEStats` 来启用扩展统计信息。如果 `SetPerTcp6ConnectionEStats` 调用失败，则对 `GetPerTcp6ConnectionEStats` 的后续调用将返回无意义的随机数据，而不是扩展的 TCP 统计信息。可以通过以管理员和普通用户身份运行以下示例来观察该示例。

示例

以下示例检索 IPv4 和 IPv6 TCP 连接的 TCP 扩展统计信息，并从返回的数据中输出值。

C++

```
// Need to link with Iphlpapi.lib and Ws2_32.lib
// Need to run as administrator

#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>
#include <winsock2.h>
#include <Ws2tcpip.h>
#include <iphlpapi.h>
#include <Tcpestats.h>
#include <stdlib.h>
#include <stdio.h>

// Need to link with Iphlpapi.lib
#pragma comment(lib, "iphlpapi.lib")

// Need to link with Ws2_32.lib
#pragma comment(lib, "ws2_32.lib")

// An array of name for the TCP_ESTATS_TYPE enum values
// The names values must match the enum values
const wchar_t* estatsTypeNames[] = {
    L"TcpConnectionEstatsSynOpts",
    L"TcpConnectionEstatsData",
    L"TcpConnectionEstatsSndCong",
    L"TcpConnectionEstatsPath",
    L"TcpConnectionEstatsSendBuff",
    L"TcpConnectionEstatsRec",
    L"TcpConnectionEstatsObsRec",
```

```

        L"TcpConnectionEstatsBandwidth",
        L"TcpConnectionEstatsFineRtt",
        L"TcpConnectionEstatsMaximum"
};

// Function prototypes

// Run tests for IPv4 or IPv4 TCP extended stats
DWORD RunEstatsTest(bool v6);

// Get an IPv4 TCP row entry
DWORD GetTcpRow(u_short localPort, u_short remotePort,
    MIB_TCP_STATE state, __out PMIB_TCPROW row);

// Get an IPv6 TCP row entry
DWORD GetTcp6Row(u_short localPort, u_short remotePort,
    MIB_TCP_STATE state, __out PMIB_TCP6ROW row);

// Enable or disable the supplied Estat type on a TCP connection
void ToggleEstat(PVOID row, TCP_ESTATS_TYPE type, bool enable, bool v6);

// Toggle all Estates for a TCP connection
void ToggleAllEstats(void* row, bool enable, bool v6);

// Dump the supplied Estate type data on the given TCP connection row
void GetAndOutputEstats(void* row, TCP_ESTATS_TYPE type, bool v6);

//
void GetAllEstats(void* row, bool v6);

// Creates a TCP server and client socket on the loopback address.
// Binds the server socket to a port.
// Establishes a client TCP connection to the server
int CreateTcpConnection(bool v6, SOCKET* serviceSocket, SOCKET*
clientSocket,
    SOCKET* acceptSocket, u_short* serverPort,
    u_short* clientPort);

//
// Entry point.
//
int __cdecl main()
{
    RunEstatsTest(FALSE);
    RunEstatsTest(TRUE);
    return (0);
}

//
// Create connect and listen sockets on loopback interface and dump all
// Estates
// types on the created TCP connections for the supplied IP address type.
//
DWORD RunEstatsTest(bool v6)
{

```

```

SOCKET serviceSocket, clientSocket, acceptSocket;
serviceSocket = clientSocket = acceptSocket = INVALID_SOCKET;
MIB_TCPROW server4ConnectRow, client4ConnectRow;
MIB_TCP6ROW server6ConnectRow, client6ConnectRow;
void* serverConnectRow, * clientConnectRow = NULL;
bool bWSAStartup = false;

char* buff = (char*)malloc(1000);
if (buff == NULL) {
    wprintf(L"\nFailed to allocate memory.");
    goto bail;
}

if (v6) {
    serverConnectRow = &server6ConnectRow;
    clientConnectRow = &client6ConnectRow;
}
else {
    serverConnectRow = &server4ConnectRow;
    clientConnectRow = &client4ConnectRow;
}

UINT winStatus;
int sockStatus;
u_short serverPort, clientPort;

// Initialize Winsock.
//
WSADATA wsaData;
winStatus = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (winStatus != ERROR_SUCCESS) {
    wprintf(L"\nFailed to open winsock. Error %d", winStatus);
    goto bail;
}

bWSAStartup = true;

// Create TCP connection on which Estats information will be collected.
// Obtain port numbers of created connections.
//
winStatus =
    CreateTcpConnection(v6, &serviceSocket, &clientSocket,
&acceptSocket,
    &serverPort, &clientPort);
if (winStatus != ERROR_SUCCESS) {
    wprintf(L"\nFailed to create TCP connection. Error %d", winStatus);
    goto bail;
}
//
// Obtain MIB_TCPROW corresponding to the TCP connection.
//
winStatus = v6 ?
    GetTcp6Row(serverPort, clientPort, MIB_TCP_STATE_ESTAB,

```

```

        (PMIB_TCP6ROW)serverConnectRow) :
    GetTcpRow(serverPort, clientPort, MIB_TCP_STATE_ESTAB,
        (PMIB_TCPROW)serverConnectRow);
if (winStatus != ERROR_SUCCESS) {
    wprintf
    (L"\nGetTcpRow failed on the server established connection with %d",
        winStatus);
    goto bail;
}

winStatus = v6 ?
    GetTcp6Row(clientPort, serverPort, MIB_TCP_STATE_ESTAB,
        (PMIB_TCP6ROW)clientConnectRow) :
    GetTcpRow(clientPort, serverPort, MIB_TCP_STATE_ESTAB,
        (PMIB_TCPROW)clientConnectRow);
if (winStatus != ERROR_SUCCESS) {
    wprintf
    (L"\nGetTcpRow failed on the client established connection with %d",
        winStatus);
    goto bail;
}
//
// Enable Estates collection and dump current stats.
//
ToggleAllEstats(serverConnectRow, TRUE, v6);
ToggleAllEstats(clientConnectRow, TRUE, v6);
wprintf(L"\n\n\nDumping Estates for server socket:\n");
GetAllEstats(serverConnectRow, v6);
wprintf(L"\n\n\nDumping Estates for client connect socket:\n");
GetAllEstats(clientConnectRow, v6);

//
// Initiate TCP data transfers to see effect on Estates counters.
//
sockStatus = send(clientSocket, buff, (int)(1000 * sizeof(char)), 0);
if (sockStatus == SOCKET_ERROR) {
    wprintf(L"\nFailed to send from client to server %d",
        WSAGetLastError());
}
else {
    sockStatus = recv(acceptSocket, buff, (int)(1000 * sizeof(char)),
0);
    if (sockStatus == SOCKET_ERROR) {
        wprintf(L"\nFailed to receive data on the server %d",
            WSAGetLastError());
    }
}

//
// Dump updated Estates and disable Estates collection.
//
wprintf
(L"\n\n\nDumping Estates for server socket after client sends data:\n");
GetAllEstats(serverConnectRow, v6);
wprintf

```

```

(L"\n\n\nDumping Estates for client socket after client sends data:\n");
GetAllEstats(clientConnectRow, v6);
ToggleAllEstats(serverConnectRow, FALSE, v6);
ToggleAllEstats(clientConnectRow, FALSE, v6);

bail:
    if (serviceSocket != INVALID_SOCKET)
        closesocket(serviceSocket);
    if (clientSocket != INVALID_SOCKET)
        closesocket(clientSocket);
    if (acceptSocket != INVALID_SOCKET)
        closesocket(acceptSocket);
    if (buff != NULL)
        free(buff);
    if (bWSAStartup)
        WSACleanup();
    return ERROR_SUCCESS;
}

int CreateTcpConnection(bool v6,
    SOCKET* serviceSocket,
    SOCKET* clientSocket,
    SOCKET* acceptSocket,
    u_short* serverPort, u_short* clientPort)
{
    INT status;
    ADDRINFOW hints, * localhost = NULL;
    const wchar_t* loopback;
    loopback = v6 ? L":1" : L"127.0.0.1";
    int aiFamily = v6 ? AF_INET6 : AF_INET;
    int nameLen = sizeof(SOCKADDR_STORAGE);

    *serviceSocket = INVALID_SOCKET;
    *clientSocket = INVALID_SOCKET;
    *acceptSocket = INVALID_SOCKET;

    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = aiFamily;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;

    status = GetAddrInfoW(loopback, L"", &hints, &localhost);
    if (status != ERROR_SUCCESS) {
        wprintf(L"\nFailed to open localhost. Error %d", status);
        goto bail;
    }

    *serviceSocket = socket(aiFamily, SOCK_STREAM, IPPROTO_TCP);
    if (*serviceSocket == INVALID_SOCKET) {
        wprintf(L"\nFailed to create server socket. Error %d",
            WSAGetLastError());
        goto bail;
    }

    *clientSocket = socket(aiFamily, SOCK_STREAM, IPPROTO_TCP);
}

```

```
if (*clientSocket == INVALID_SOCKET) {
    wprintf(L"\nFailed to create client socket. Error %d",
           WSAGetLastError());
    goto bail;
}

status =
    bind(*serviceSocket, localhost->ai_addr, (int)localhost-
>ai_addrlen);
if (status == SOCKET_ERROR) {
    wprintf(L"\nFailed to bind server socket to loopback. Error %d",
           WSAGetLastError());
    goto bail;
}

if (localhost != NULL) {
    FreeAddrInfoW(localhost);
    localhost = NULL;
}

SOCKADDR_STORAGE serverSockName, clientSockName;
status = getsockname(*serviceSocket,
    (sockaddr*)&serverSockName, &nameLen);
if (status == SOCKET_ERROR) {
    wprintf(L"\ngetsockname failed %d", WSAGetLastError());
    goto bail;
}
if (v6) {
    *serverPort = ((sockaddr_in6*)(&serverSockName))->sin6_port;
}
else {
    *serverPort = ((sockaddr_in*)(&serverSockName))->sin_port;
}

status = listen(*serviceSocket, SOMAXCONN);
if (status == SOCKET_ERROR) {
    wprintf(L"\nFailed to listen on server socket. Error %d",
           WSAGetLastError());
    goto bail;
}

status =
    connect(*clientSocket, (sockaddr*)&serverSockName,
            (int)sizeof(SOCKADDR_STORAGE));
if (status == SOCKET_ERROR) {
    wprintf(L"\nCould not connect client and server sockets %d",
           WSAGetLastError());
    goto bail;
}

status = getsockname(*clientSocket,
    (sockaddr*)&clientSockName, &nameLen);
if (status == SOCKET_ERROR) {
    wprintf(L"\ngetsockname failed %d", WSAGetLastError());
    goto bail;
}
```

```

    }

    if (v6) {
        *clientPort = ((sockaddr_in6*)(&clientSockName))->sin6_port;
    }
    else {
        *clientPort = ((sockaddr_in*)(&clientSockName))->sin_port;
    }

    *acceptSocket = accept(*serviceSocket, NULL, NULL);
    if (*acceptSocket == INVALID_SOCKET) {
        wprintf(L"\nFailed to accept socket connection %d",
WSAGetLastError());
        goto bail;
    }

    return ERROR_SUCCESS;

bail:
    if (localhost != NULL)
        FreeAddrInfoW(localhost);

    if (*serviceSocket != INVALID_SOCKET) {
        closesocket(*serviceSocket);
        *serviceSocket = INVALID_SOCKET;
    }

    if (*clientSocket != INVALID_SOCKET) {
        closesocket(*clientSocket);
        *clientSocket = INVALID_SOCKET;
    }

    if (*acceptSocket != INVALID_SOCKET) {
        closesocket(*acceptSocket);
        *acceptSocket = INVALID_SOCKET;
    }

    return status;
}

void GetAllEstats(void* row, bool v6)
{
    GetAndOutputEstats(row, TcpConnectionEstatsSynOpts, v6);
    GetAndOutputEstats(row, TcpConnectionEstatsData, v6);
    GetAndOutputEstats(row, TcpConnectionEstatsSndCong, v6);
    GetAndOutputEstats(row, TcpConnectionEstatsPath, v6);
    GetAndOutputEstats(row, TcpConnectionEstatsSendBuff, v6);
    GetAndOutputEstats(row, TcpConnectionEstatsRec, v6);
    GetAndOutputEstats(row, TcpConnectionEstatsObsRec, v6);
    GetAndOutputEstats(row, TcpConnectionEstatsBandwidth, v6);
    GetAndOutputEstats(row, TcpConnectionEstatsFineRtt, v6);
}

//
// Returns a MIB_TCPROW corresponding to the local port, remote port and state

```

```
// filter parameters.
//
DWORD
GetTcpRow(u_short localPort,
    u_short remotePort, MIB_TCP_STATE state, __out PMIB_TCPROW row)
{
    PMIB_TCPTABLE tcpTable = NULL;
    PMIB_TCPROW tcpRowIt = NULL;

    DWORD status, size = 0, i;
    bool connectionFound = FALSE;

    status = GetTcpTable(tcpTable, &size, TRUE);
    if (status != ERROR_INSUFFICIENT_BUFFER) {
        return status;
    }

    tcpTable = (PMIB_TCPTABLE)malloc(size);
    if (tcpTable == NULL) {
        return ERROR_OUTOFMEMORY;
    }

    status = GetTcpTable(tcpTable, &size, TRUE);
    if (status != ERROR_SUCCESS) {
        free(tcpTable);
        return status;
    }

    for (i = 0; i < tcpTable->dwNumEntries; i++) {
        tcpRowIt = &tcpTable->table[i];
        if (tcpRowIt->dwLocalPort == (DWORD)localPort &&
            tcpRowIt->dwRemotePort == (DWORD)remotePort &&
            tcpRowIt->State == state) {
            connectionFound = TRUE;
            *row = *tcpRowIt;
            break;
        }
    }

    free(tcpTable);

    if (connectionFound) {
        return ERROR_SUCCESS;
    }
    else {
        return ERROR_NOT_FOUND;
    }
}

//
// Returns a MIB_TCP6ROW corresponding to the local port, remote port and
// state
// filter parameters. This is a v6 equivalent of the GetTcpRow function.
//
DWORD
```

```

GetTcp6Row(u_short localPort,
           u_short remotePort, MIB_TCP_STATE state, __out PMIB_TCP6ROW row)
{
    PMIB_TCP6TABLE tcp6Table = NULL;
    PMIB_TCP6ROW tcp6RowIt = NULL;

    DWORD status, size = 0, i;
    bool connectionFound = FALSE;

    status = GetTcp6Table(tcp6Table, &size, TRUE);
    if (status != ERROR_INSUFFICIENT_BUFFER) {
        return status;
    }

    tcp6Table = (PMIB_TCP6TABLE)malloc(size);
    if (tcp6Table == NULL) {
        return ERROR_OUTOFMEMORY;
    }

    status = GetTcp6Table(tcp6Table, &size, TRUE);
    if (status != ERROR_SUCCESS) {
        free(tcp6Table);
        return status;
    }

    for (i = 0; i < tcp6Table->dwNumEntries; i++) {
        tcp6RowIt = &tcp6Table->table[i];
        if (tcp6RowIt->dwLocalPort == (DWORD)localPort &&
            tcp6RowIt->dwRemotePort == (DWORD)remotePort &&
            tcp6RowIt->State == state) {
            connectionFound = TRUE;
            *row = *tcp6RowIt;
            break;
        }
    }

    free(tcp6Table);

    if (connectionFound) {
        return ERROR_SUCCESS;
    }
    else {
        return ERROR_NOT_FOUND;
    }
}

// 
// Enable or disable the supplied Estat type on a TCP connection.
//
void ToggleEstat(PVOID row, TCP_ESTATS_TYPE type, bool enable, bool v6)
{
    TCP_BOOLEAN_OPTIONAL operation =
        enable ? TcpBoolOptEnabled : TcpBoolOptDisabled;
    ULONG status, size = 0;
    PUCHAR rw = NULL;
}

```

```

TCP_ESTATS_DATA_RW_v0 dataRw;
TCP_ESTATS SND_CONG_RW_v0 sndRw;
TCP_ESTATS PATH_RW_v0 pathRw;
TCP_ESTATS SEND_BUFF_RW_v0 sendBuffRw;
TCP_ESTATS REC_RW_v0 recRw;
TCP_ESTATS OBS_REC_RW_v0 obsRecRw;
TCP_ESTATS BANDWIDTH_RW_v0 bandwidthRw;
TCP_ESTATS FINE_RTT_RW_v0 fineRttRw;

switch (type) {
case TcpConnectionEstatsData:
    dataRw.EnableCollection = enable;
    rw = (P UCHAR)&dataRw;
    size = sizeof(TCP_ESTATS_DATA_RW_v0);
    break;

case TcpConnectionEstatsSndCong:
    sndRw.EnableCollection = enable;
    rw = (P UCHAR)&sndRw;
    size = sizeof(TCP_ESTATS SND_CONG_RW_v0);
    break;

case TcpConnectionEstatsPath:
    pathRw.EnableCollection = enable;
    rw = (P UCHAR)&pathRw;
    size = sizeof(TCP_ESTATS PATH_RW_v0);
    break;

case TcpConnectionEstatsSendBuff:
    sendBuffRw.EnableCollection = enable;
    rw = (P UCHAR)&sendBuffRw;
    size = sizeof(TCP_ESTATS SEND_BUFF_RW_v0);
    break;

case TcpConnectionEstatsRec:
    recRw.EnableCollection = enable;
    rw = (P UCHAR)&recRw;
    size = sizeof(TCP_ESTATS REC_RW_v0);
    break;

case TcpConnectionEstatsObsRec:
    obsRecRw.EnableCollection = enable;
    rw = (P UCHAR)&obsRecRw;
    size = sizeof(TCP_ESTATS OBS_REC_RW_v0);
    break;

case TcpConnectionEstatsBandwidth:
    bandwidthRw.EnableCollectionInbound = operation;
    bandwidthRw.EnableCollectionOutbound = operation;
    rw = (P UCHAR)&bandwidthRw;
    size = sizeof(TCP_ESTATS BANDWIDTH_RW_v0);
    break;

case TcpConnectionEstatsFineRtt:
    fineRttRw.EnableCollection = enable;
}

```

```

        rw = (PUCHAR)&fineRttRw;
        size = sizeof(TCP_ESTATS_FINE_RTT_RW_v0);
        break;

    default:
        return;
        break;
    }

    if (v6) {
        status = SetPerTcp6ConnectionEStats((PMIB_TCP6ROW)row, type,
            rw, 0, size, 0);
    }
    else {
        status = SetPerTcpConnectionEStats((PMIB_TCPROW)row, type,
            rw, 0, size, 0);
    }

    if (status != NO_ERROR) {
        if (v6)
            wprintf(L"\nSetPerTcp6ConnectionEStats %s %s failed. status =
%d",
                estatsTypeNames[type], enable ? L"enabled" : L"disabled",
                status);
        else
            wprintf(L"\nSetPerTcpConnectionEStats %s %s failed. status =
%d",
                estatsTypeNames[type], enable ? L"enabled" : L"disabled",
                status);
    }
}

// // Toggle all Estats for a TCP connection.
// void ToggleAllEstats(void* row, bool enable, bool v6)
{
    ToggleEstat(row, TcpConnectionEstatsData, enable, v6);
    ToggleEstat(row, TcpConnectionEstatsSndCong, enable, v6);
    ToggleEstat(row, TcpConnectionEstatsPath, enable, v6);
    ToggleEstat(row, TcpConnectionEstatsSendBuff, enable, v6);
    ToggleEstat(row, TcpConnectionEstatsRec, enable, v6);
    ToggleEstat(row, TcpConnectionEstatsObsRec, enable, v6);
    ToggleEstat(row, TcpConnectionEstatsBandwidth, enable, v6);
    ToggleEstat(row, TcpConnectionEstatsFineRtt, enable, v6);
}

// // Call GetPerTcp6ConnectionEStats or GetPerTcpConnectionEStats.
// ULONG GetConnectionEStats(void* row, TCP_ESTATS_TYPE type, PUCHAR rw, ULONG
rwSize, bool v6, PUCHAR ros, ULONG rosSize, PUCHAR rod, ULONG rodSize)
{
    if (v6) {
        return GetPerTcp6ConnectionEStats((PMIB_TCP6ROW)row,

```

```

        type,
        rw, 0, rwSize,
        ros, 0, rosSize,
        rod, 0, rodSize);
    }
    else {
        return GetPerTcpConnectionEStats((PMIB_TCPROW)row,
            type,
            rw, 0, rwSize,
            ros, 0, rosSize,
            rod, 0, rodSize);
    }
}

//  

// Dump the supplied Estate type on the given TCP connection row.  

//  

void GetAndOutputEStats(void* row, TCP_ESTATS_TYPE type, bool v6)
{
    ULONG rosSize = 0, rodSize = 0;
    ULONG winStatus;
    PUCHAR ros = NULL, rod = NULL;

    PTCP_ESTATS_SYN_OPTS_ROS_v0 synOptsRos = { 0 };
    PTCP_ESTATS_DATA_ROD_v0 dataRod = { 0 };
    PTCP_ESTATS SND_CONG_ROD_v0 sndCongRod = { 0 };
    PTCP_ESTATS SND_CONG_ROS_v0 sndCongRos = { 0 };
    PTCP_ESTATS_PATH_ROD_v0 pathRod = { 0 };
    PTCP_ESTATS_SEND_BUFF_ROD_v0 sndBuffRod = { 0 };
    PTCP_ESTATS_REC_ROD_v0 recRod = { 0 };
    PTCP_ESTATS_OBS_REC_ROD_v0 obsRecRod = { 0 };
    PTCP_ESTATS_BANDWIDTH_ROD_v0 bandwidthRod = { 0 };
    PTCP_ESTATS_FINE_RTT_ROD_v0 fineRttRod = { 0 };

    switch (type) {
        case TcpConnectionEstatsSynOpts:
            rosSize = sizeof(TCP_ESTATS_SYN_OPTS_ROS_v0);
            break;

        case TcpConnectionEstatsData:
            rodSize = sizeof(TCP_ESTATS_DATA_ROD_v0);
            break;

        case TcpConnectionEstatsSndCong:
            rodSize = sizeof(TCP_ESTATS SND_CONG_ROD_v0);
            rosSize = sizeof(TCP_ESTATS SND_CONG_ROS_v0);
            break;

        case TcpConnectionEstatsPath:
            rodSize = sizeof(TCP_ESTATS_PATH_ROD_v0);
            break;

        case TcpConnectionEstatsSendBuff:
            rodSize = sizeof(TCP_ESTATS_SEND_BUFF_ROD_v0);
            break;
    }
}

```

```

case TcpConnectionEstatsRec:
    rodSize = sizeof(TCP_ESTATS_REC_ROD_v0);
    break;

case TcpConnectionEstatsObsRec:
    rodSize = sizeof(TCP_ESTATS_OBS_REC_ROD_v0);
    break;

case TcpConnectionEstatsBandwidth:
    rodSize = sizeof(TCP_ESTATS_BANDWIDTH_ROD_v0);
    break;

case TcpConnectionEstatsFineRtt:
    rodSize = sizeof(TCP_ESTATS_FINE_RTT_ROD_v0);
    break;

default:
    wprintf(L"\nCannot get type %d", (int)type);
    return;
    break;
}

if (rossize != 0) {
    ros = (P UCHAR)malloc(rossize);
    if (ros == NULL) {
        wprintf(L"\nOut of memory");
        return;
    }
    else
        memset(ros, 0, rossize); // zero the buffer
}
if (rodsize != 0) {
    rod = (P UCHAR)malloc(rodsize);
    if (rod == NULL) {
        free(ros);
        wprintf(L"\nOut of memory");
        return;
    }
    else
        memset(rod, 0, rodsize); // zero the buffer
}

TCP_ESTATS_DATA_RW_v0 dataRw = { 0 };
TCP_ESTATS SND_CONG_RW_v0 sndCongRw = { 0 };
TCP_ESTATS PATH_RW_v0 pathRw = { 0 };
TCP_ESTATS SEND_BUFF_RW_v0 sndBuffRw = { 0 };
TCP_ESTATS REC_RW_v0 recRw = { 0 };
TCP_ESTATS OBS_REC_RW_v0 obsRecRw = { 0 };
TCP_ESTATS BANDWIDTH_RW_v0 bandwidthRw = { 0 };
TCP_ESTATS FINE_RTT_RW_v0 fineRttRw = { 0 };
BOOLEAN RwEnableCollection{ FALSE };

switch (type) {
case TcpConnectionEstatsData:

```

```

        winStatus = GetConnectionEStats(row, type, (P UCHAR)&dataRw,
sizeof(TCP_ESTATS_DATA_RW_v0), v6, ros, rosSize, rod, rodSize);
        RwEnableCollection = dataRw.EnableCollection;
        break;

    case TcpConnectionEStatsSndCong:
        winStatus = GetConnectionEStats(row, type, (P UCHAR)&sndCongRw,
sizeof(TCP_ESTATS SND CONG RW_v0), v6, ros, rosSize, rod, rodSize);
        RwEnableCollection = sndCongRw.EnableCollection;
        break;

    case TcpConnectionEStatsPath:
        winStatus = GetConnectionEStats(row, type, (P UCHAR)&pathRw,
sizeof(TCP_ESTATS_PATH_RW_v0), v6, ros, rosSize, rod, rodSize);
        RwEnableCollection = pathRw.EnableCollection;
        break;

    case TcpConnectionEStatsSendBuff:
        winStatus = GetConnectionEStats(row, type, (P UCHAR)&sndBuffRw,
sizeof(TCP_ESTATS SEND BUFF RW_v0), v6, ros, rosSize, rod, rodSize);
        RwEnableCollection = sndBuffRw.EnableCollection;
        break;

    case TcpConnectionEStatsRec:
        winStatus = GetConnectionEStats(row, type, (P UCHAR)&recRw,
sizeof(TCP_ESTATS REC RW_v0), v6, ros, rosSize, rod, rodSize);
        RwEnableCollection = recRw.EnableCollection;
        break;

    case TcpConnectionEStatsObsRec:
        winStatus = GetConnectionEStats(row, type, (P UCHAR)&obsRecRw,
sizeof(TCP_ESTATS OBS REC RW_v0), v6, ros, rosSize, rod, rodSize);
        RwEnableCollection = obsRecRw.EnableCollection;
        break;

    case TcpConnectionEStatsBandwidth:
        winStatus = GetConnectionEStats(row, type, (P UCHAR)&bandwidthRw,
sizeof(TCP_ESTATS BANDWIDTH RW_v0), v6, ros, rosSize, rod, rodSize);
        RwEnableCollection = bandwidthRw.EnableCollectionOutbound &&
bandwidthRw.EnableCollectionInbound;
        break;

    case TcpConnectionEStatsFineRtt:
        winStatus = GetConnectionEStats(row, type, (P UCHAR)&fineRttRw,
sizeof(TCP_ESTATS FINE RTT RW_v0), v6, ros, rosSize, rod, rodSize);
        RwEnableCollection = fineRttRw.EnableCollection;
        break;

    default:
        winStatus = GetConnectionEStats(row, type, NULL, v6, 0, ros,
rosSize, rod, rodSize);
        break;
    }

    if (!RwEnableCollection) {

```

```

    if (v6)
        wprintf(L"\nGetPerTcp6ConnectionEStats %s failed.
Rw.EnableCollection == FALSE", estatsTypeNames[type]);
    else
        wprintf(L"\nGetPerTcpConnectionEStats %s failed.
Rw.EnableCollection == FALSE", estatsTypeNames[type]);
    return;
}

if (winStatus != NO_ERROR) {
    if (v6)
        wprintf(L"\nGetPerTcp6ConnectionEStats %s failed. status = %d",
            estatsTypeNames[type],
            winStatus);
    else
        wprintf(L"\nGetPerTcpConnectionEStats %s failed. status = %d",
            estatsTypeNames[type],
            winStatus);
}
else {
    switch (type) {
        case TcpConnectionEstatsSynOpts:
            synOptsRos = (PTCP_ESTATS_SYN_OPTS_ROS_v0)ros;
            wprintf(L"\nSyn Opts");
            wprintf(L"\nActive Open:    %s",
                synOptsRos->ActiveOpen ? L"Yes" : L"No");
            wprintf(L"\nMss Received:   %u", synOptsRos->MssRcvd);
            wprintf(L"\nMss Sent         %u", synOptsRos->MssSent);
            break;

        case TcpConnectionEstatsData:
            dataRod = (PTCP_ESTATS_DATA_ROD_v0)rod;
            wprintf(L"\n\nData");
            wprintf(L"\nBytes Out:     %lu", dataRod->DataBytesOut);
            wprintf(L"\nSegs Out:      %lu", dataRod->DataSegsOut);
            wprintf(L"\nBytes In:      %lu", dataRod->DataBytesIn);
            wprintf(L"\nSegs In:       %lu", dataRod->DataSegsIn);
            wprintf(L"\nSegs Out:      %u", dataRod->SegsOut);
            wprintf(L"\nSegs In:       %u", dataRod->SegsIn);
            wprintf(L"\nSoft Errors:   %u", dataRod->SoftErrors);
            wprintf(L"\nSoft Error Reason: %u", dataRod->SoftErrorReason);
            wprintf(L"\nSnd Una:       %u", dataRod->SndUna);
            wprintf(L"\nSnd Nxt:       %u", dataRod->SndNxt);
            wprintf(L"\nSnd Max:       %u", dataRod->SndMax);
            wprintf(L"\nBytes Acked:   %lu", dataRod->ThruBytesAcked);
            wprintf(L"\nRcv Nxt:       %u", dataRod->RcvNxt);
            wprintf(L"\nBytes Rcv:     %lu", dataRod->ThruBytesReceived);
            break;

        case TcpConnectionEstatsSndCong:
            sndCongRod = (PTCP_ESTATS SND_CONG_ROD_v0)rod;
            sndCongRos = (PTCP_ESTATS SND_CONG_ROS_v0)ros;
            wprintf(L"\n\nSnd Cong");
            wprintf(L"\nTrans Rwin:    %u", sndCongRod->SndLimTransRwin);
            wprintf(L"\nLim Time Rwin: %u", sndCongRod->SndLimTimeRwin);
    }
}

```

```

        wprintf(L"\nLim Bytes Rwin: %u", sndCongRod->SndLimBytesRwin);
        wprintf(L"\nLim Trans Cwnd: %u", sndCongRod->SndLimTransCwnd);
        wprintf(L"\nLim Time Cwnd: %u", sndCongRod->SndLimTimeCwnd);
        wprintf(L"\nLim Bytes Cwnd: %u", sndCongRod->SndLimBytesCwnd);
        wprintf(L"\nLim Trans Snd: %u", sndCongRod->SndLimTransSnd);
        wprintf(L"\nLim Time Snd: %u", sndCongRod->SndLimTimeSnd);
        wprintf(L"\nLim Bytes Snd: %u", sndCongRod->SndLimBytesSnd);
        wprintf(L"\nSlow Start: %u", sndCongRod->SlowStart);
        wprintf(L"\nCong Avoid: %u", sndCongRod->CongAvoid);
        wprintf(L"\nOther Reductions: %u", sndCongRod->OtherReductions);
        wprintf(L"\nCur Cwnd: %u", sndCongRod->CurCwnd);
        wprintf(L"\nMax Ss Cwnd: %u", sndCongRod->MaxSsCwnd);
        wprintf(L"\nMax Ca Cwnd: %u", sndCongRod->MaxCaCwnd);
        wprintf(L"\nCur Ss Thresh: 0x%x (%u)", sndCongRod-
>CurSsthresh,
                sndCongRod->CurSsthresh);
        wprintf(L"\nMax Ss Thresh: 0x%x (%u)", sndCongRod-
>MaxSsthresh,
                sndCongRod->MaxSsthresh);
        wprintf(L"\nMin Ss Thresh: 0x%x (%u)", sndCongRod-
>MinSsthresh,
                sndCongRod->MinSsthresh);
        wprintf(L"\nLim Cwnd: 0x%x (%u)", sndCongRos->LimCwnd,
                sndCongRos->LimCwnd);
        break;

    case TcpConnectionEstatsPath:
        pathRod = (PTCP_ESTATS_PATH_ROD_v0)rod;
        wprintf(L"\n\nPath");
        wprintf(L"\nFast Retran: %u", pathRod->FastRetran);
        wprintf(L"\nTimeouts: %u", pathRod->Timeouts);
        wprintf(L"\nSubsequent Timeouts: %u", pathRod-
>SubsequentTimeouts);
        wprintf(L"\nCur Timeout Count: %u", pathRod->CurTimeoutCount);
        wprintf(L"\nAbrupt Timeouts: %u", pathRod->AbruptTimeouts);
        wprintf(L"\nPkts Retrans: %u", pathRod->PktsRetrans);
        wprintf(L"\nBytes Retrans: %u", pathRod->BytesRetrans);
        wprintf(L"\nDup Acks In: %u", pathRod->DupAcksIn);
        wprintf(L"\nSacksRcvd: %u", pathRod->SacksRcvd);
        wprintf(L"\nSack Blocks Rcvd: %u", pathRod->SackBlocksRcvd);
        wprintf(L"\nCong Signals: %u", pathRod->CongSignals);
        wprintf(L"\nPre Cong Sum Cwnd: %u", pathRod->PreCongSumCwnd);
        wprintf(L"\nPre Cong Sum Rtt: %u", pathRod->PreCongSumRtt);
        wprintf(L"\nPost Cong Sum Rtt: %u", pathRod->PostCongSumRtt);
        wprintf(L"\nPost Cong Count Rtt: %u", pathRod-
>PostCongCountRtt);
        wprintf(L"\nEcn Signals: %u", pathRod->EcnSignals);
        wprintf(L"\nEce Rcvd: %u", pathRod->EceRcvd);
        wprintf(L"\nSend Stall: %u", pathRod->SendStall);
        wprintf(L"\nQuench Rcvd: %u", pathRod->QuenchRcvd);
        wprintf(L"\nRetran Thresh: %u", pathRod->RetranThresh);
        wprintf(L"\nSnd Dup Ack Episodes: %u", pathRod-
>SndDupAckEpisodes);
        wprintf(L"\nSum Bytes Reordered: %u", pathRod-
>SumBytesReordered);

```

```

        wprintf(L"\nNon Recov Da:           %u", pathRod->NonRecovDa);
        wprintf(L"\nNon Recov Da Episodes: %u", pathRod-
>NonRecovDaEpisodes);
        wprintf(L"\nAck After Fr:         %u", pathRod->AckAfterFr);
        wprintf(L"\nDsack Dups:           %u", pathRod->DsackDups);
        wprintf(L"\nSample Rtt:             0x%u (%u)", pathRod->SampleRtt,
                pathRod->SampleRtt);
        wprintf(L"\nSmoothed Rtt:          %u", pathRod->SmoothedRtt);
        wprintf(L"\nRtt Var:                %u", pathRod->RttVar);
        wprintf(L"\nMax Rtt:                %u", pathRod->MaxRtt);
        wprintf(L"\nMin Rtt:                0x%u (%u)", pathRod->MinRtt,
                pathRod->MinRtt);
        wprintf(L"\nSum Rtt:                %u", pathRod->SumRtt);
        wprintf(L"\nCount Rtt:              %u", pathRod->CountRtt);
        wprintf(L"\nCur Rto:                %u", pathRod->CurRto);
        wprintf(L"\nMax Rto:                %u", pathRod->MaxRto);
        wprintf(L"\nMin Rto:                %u", pathRod->MinRto);
        wprintf(L"\nCur MSS:                %u", pathRod->CurMss);
        wprintf(L"\nMax MSS:                %u", pathRod->MaxMss);
        wprintf(L"\nMin MSS:                %u", pathRod->MinMss);
        wprintf(L"\nSpurious Rto:            %u", pathRod-
>SpuriousRtoDetections);
        break;

    case TcpConnectionEstatsSendBuff:
        sndBuffRod = (PTCP_ESTATS_SEND_BUFF_ROD_v0)rod;
        wprintf(L"\n\nSend Buff");
        wprintf(L"\nCur Retx Queue:   %u", sndBuffRod->CurRetxQueue);
        wprintf(L"\nMax Retx Queue:   %u", sndBuffRod->MaxRetxQueue);
        wprintf(L"\nCur App W Queue: %u", sndBuffRod->CurAppWQueue);
        wprintf(L"\nMax App W Queue: %u", sndBuffRod->MaxAppWQueue);
        break;

    case TcpConnectionEstatsRec:
        recRod = (PTCP_ESTATS_REC_ROD_v0)rod;
        wprintf(L"\n\nRec");
        wprintf(L"\nCur Rwin Sent:    0x%u (%u)", recRod->CurRwinSent,
                recRod->CurRwinSent);
        wprintf(L"\nMax Rwin Sent:    0x%u (%u)", recRod->MaxRwinSent,
                recRod->MaxRwinSent);
        wprintf(L"\nMin Rwin Sent:    0x%u (%u)", recRod->MinRwinSent,
                recRod->MinRwinSent);
        wprintf(L"\nLim Rwin:         0x%u (%u)", recRod->LimRwin,
                recRod->LimRwin);
        wprintf(L"\nDup Ack Episodes: %u", recRod->DupAckEpisodes);
        wprintf(L"\nDup Ack Out:       %u", recRod->DupAcksOut);
        wprintf(L"\nCe Rcvd:           %u", recRod->CeRcvd);
        wprintf(L"\nEcn Send:           %u", recRod->EcnSent);
        wprintf(L"\nEcn Nonces Rcvd:  %u", recRod->EcnNoncesRcvd);
        wprintf(L"\nCur Reasm Queue:  %u", recRod->CurReasmQueue);
        wprintf(L"\nMax Reasm Queue:  %u", recRod->MaxReasmQueue);
        wprintf(L"\nCur App R Queue:  %u", recRod->CurAppRQueue);
        wprintf(L"\nMax App R Queue:  %u", recRod->MaxAppRQueue);
        wprintf(L"\nWin Scale Sent:    0x%.2x", recRod->WinScaleSent);
        break;

```

```

    case TcpConnectionEstatsObsRec:
        obsRecRod = (PTCP_ESTATS_OBS_REC_ROD_v0)rod;
        wprintf(L"\n\nObs Rec");
        wprintf(L"\nCur Rwin Rcvd: 0x%u (%u)", obsRecRod->CurRwinRcvd,
               obsRecRod->CurRwinRcvd);
        wprintf(L"\nMax Rwin Rcvd: 0x%u (%u)", obsRecRod->MaxRwinRcvd,
               obsRecRod->MaxRwinRcvd);
        wprintf(L"\nMin Rwin Rcvd: 0x%u (%u)", obsRecRod->MinRwinRcvd,
               obsRecRod->MinRwinRcvd);
        wprintf(L"\nWin Scale Rcvd: 0x%u (%u)", obsRecRod-
>WinScaleRcvd,
               obsRecRod->WinScaleRcvd);
        break;

    case TcpConnectionEstatsBandwidth:
        bandwidthRod = (PTCP_ESTATS_BANDWIDTH_ROD_v0)rod;
        wprintf(L"\n\nBandwidth");
        wprintf(L"\nOutbound Bandwidth: %lu",
               bandwidthRod->OutboundBandwidth);
        wprintf(L"\nInbound Bandwidth: %lu", bandwidthRod-
>InboundBandwidth);
        wprintf(L"\nOutbound Instability: %lu",
               bandwidthRod->OutboundInstability);
        wprintf(L"\nInbound Instability: %lu",
               bandwidthRod->InboundInstability);
        wprintf(L"\nOutbound Bandwidth Peaked: %s",
               bandwidthRod->OutboundBandwidthPeaked ? L"Yes" : L"No");
        wprintf(L"\nInbound Bandwidth Peaked: %s",
               bandwidthRod->InboundBandwidthPeaked ? L"Yes" : L"No");
        break;

    case TcpConnectionEstatsFineRtt:
        fineRttRod = (PTCP_ESTATS_FINE_RTT_ROD_v0)rod;
        wprintf(L"\n\nFine RTT");
        wprintf(L"\nRtt Var: %u", fineRttRod->RttVar);
        wprintf(L"\nMax Rtt: %u", fineRttRod->MaxRtt);
        wprintf(L"\nMin Rtt: 0x%u (%u)", fineRttRod->MinRtt,
               fineRttRod->MinRtt);
        wprintf(L"\nSum Rtt: %u", fineRttRod->SumRtt);
        break;

    default:
        wprintf(L"\nCannot get type %d", type);
        break;
    }
}

free(ros);
free(rod);
}

```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetPerTcpConnectionEStats](#)

[GetTcp6Table](#)

[MIB_TCP6ROW](#)

[MIB_TCP6TABLE](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_BANDWIDTH_ROD_v0](#)

[TCP_ESTATS_BANDWIDTH_RW_v0](#)

[TCP_ESTATS_DATA_ROD_v0](#)

[TCP_ESTATS_DATA_RW_v0](#)

[TCP_ESTATS_FINE_RTT_ROD_v0](#)

[TCP_ESTATS_FINE_RTT_RW_v0](#)

[TCP_ESTATS_OBS_REC_ROD_v0](#)

[TCP_ESTATS_OBS_REC_RW_v0](#)

[TCP_ESTATS_PATH_ROD_v0](#)

TCP_ESTATS_PATH_RW_v0

TCP_ESTATS_REC_ROD_v0

TCP_ESTATS_REC_RW_v0

TCP_ESTATS_SEND_BUFF_ROD_v0

TCP_ESTATS_SEND_BUFF_RW_v0

TCP_ESTATS SND CONG ROD_v0

TCP_ESTATS SND CONG ROS_v0

TCP_ESTATS SND CONG RW_v0

TCP_ESTATS_SYN_OPTS_ROS_v0

TCP_ESTATS_TYPE

TCP_SOFT_ERROR

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

GetPerTcpConnectionEStats 函数 (iphlpapi.h)

项目2023/08/24

GetPerTcpConnectionEStats 函数检索 IPv4 TCP 连接的扩展统计信息。

语法

C++

```
IPHLAPI_DLL_LINKAGE ULONG GetPerTcpConnectionEStats(
    PMIB_TCPROW      Row,
    TCP_ESTATS_TYPE  EstatsType,
    [out] PUCHAR      Rw,
    ULONG            RwVersion,
    ULONG            RwSize,
    [out] PUCHAR      Ros,
    ULONG            RosVersion,
    ULONG            RosSize,
    [out] PUCHAR      Rod,
    ULONG            RodVersion,
    ULONG            RodSize
);
```

parameters

Row

指向 IPv4 TCP 连接的 [MIB_TCPROW](#) 结构的指针。

EstatsType

请求的 TCP 扩展统计信息的类型。如果调用成功，此参数确定在 *Rw*、*Rod* 和 *Ros* 参数中返回的信息的数据和格式。

此参数可以是 *Tcpestats.h* 头文件中定义的 [TCP_ESTATS_TYPE](#) 枚举类型的值之一。

值	含义
TcpConnectionEStatsSynOpts	此值请求 TCP 连接的 SYN 交换信息。 此枚举值仅提供只读静态信息。

	<p>如果 <i>Ros</i> 参数不为 NULL 且函数成功, 则 <i>Ros</i> 参数指向的缓冲区应包含 TCP_ESTATS_SYN_OPTS_ROS_v0 结构。</p>
TcpConnectionEstatsData	<p>此值请求 TCP 连接的扩展数据传输信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_DATA_RW_v0 结构。</p> <p>如果为此 TCP 连接启用了扩展数据传输信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_DATA_ROD_v0 结构。</p>
TcpConnectionEstatsSndCong	<p>此值请求 TCP 连接的发送方拥塞。 (只读静态、只读动态和读/写信息的所有三种类型的信息都可用于此枚举值。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS SND_CONG_RW_v0 结构。</p> <p>如果 <i>Ros</i> 参数不为 NULL 且函数成功, 则 <i>Ros</i> 参数指向的缓冲区应包含 TCP_ESTATS SND_CONG_ROS_v0 结构。</p> <p>如果为此 TCP 连接启用了发送方拥塞信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS SND_CONG_ROD_v0 结构。</p>
TcpConnectionEstatsPath	<p>此值请求 TCP 连接的扩展路径度量信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_PATH_RW_v0 结构。</p> <p>如果为此 TCP 连接启用了扩展路径度量信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_PATH_ROD_v0 结构。</p>
TcpConnectionEstatsSendBuff	<p>此值请求 TCP 连接的扩展输出队列信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_SEND_BUFF_RW_v0 结构。</p> <p>如果为此 TCP 连接启用了扩展输出队列信息, 则 <i>Rod</i> 参数不为 NULL, 并且函数成功, <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_SEND_BUFF_ROD_v0 结构。</p>
TcpConnectionEstatsRec	<p>此值请求 TCP 连接的扩展本地接收器信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_REC_RW_v0 结构。</p>

	<p>如果为此 TCP 连接启用了扩展的本地接收器信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, 则 <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_REC_ROD_v0 结构。</p>
TcpConnectionEstatsObsRec	<p>此值请求 TCP 连接的扩展远程接收方信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_OBS_REC_RW_v0 结构。</p>
TcpConnectionEstatsBandwidth	<p>此值请求带宽上的 TCP 连接的带宽估计统计信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_BANDWIDTH_RW_v0 结构。</p>
TcpConnectionEstatsFineRtt	<p>此值请求 TCP 连接的精细往返时间 (RTT) 估计统计信息。 此枚举值只能使用只读动态信息和读/写信息。</p> <p>如果 <i>Rw</i> 参数不为 NULL 且函数成功, 则 <i>Rw</i> 参数指向的缓冲区应包含 TCP_ESTATS_FINE_RTT_RW_v0 结构。</p> <p>如果为此 TCP 连接启用了细粒度 RTT 估计统计信息, <i>Rod</i> 参数不为 NULL, 并且函数成功, 则 <i>Rod</i> 参数指向的缓冲区应包含 TCP_ESTATS_FINE_RTT_ROD_v0 结构。</p>

[out] *Rw*

指向用于接收读/写信息的缓冲区的指针。如果应用程序不想检索 TCP 连接的读/写信息, 此参数可能是 **NULL** 指针。

RwVersion

请求的读/写信息的版本。当前支持的值为 0 的版本。

RwSize

Rw 参数指向的缓冲区的大小 (以字节为单位)。

[out] *Ros*

指向用于接收只读静态信息的缓冲区的指针。如果应用程序不想检索 TCP 连接的只读静态信息，此参数可能是 **NULL** 指针。

RosVersion

请求的只读静态信息的版本。当前支持的值为 0 的版本。

RosSize

Ros 参数指向的缓冲区的大小（以字节为单位）。

[out] Rod

指向用于接收只读动态信息的缓冲区的指针。如果应用程序不想检索 TCP 连接的只读动态信息，此参数可能是 **NULL** 指针。

RodVersion

请求的只读动态信息的版本。当前支持的值为 0 的版本。

RodSize

Rod 参数指向的缓冲区的大小（以字节为单位）。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	传递给函数的缓冲区太小。如果 <i>Rw</i> 、 <i>Ros</i> 或 <i>Rod</i> 参数指向的缓冲区不够大，无法接收数据，则返回此错误。如果 <i>Rw</i> 、 <i>Ros</i> 或 <i>Rod</i> 参数指向的给定缓冲区之一为 NULL ，但在关联的 <i>RwSize</i> 、 <i>RosSize</i> 或 <i>RodSize</i> 中指定了长度，则也会返回此错误。 此错误值在 Windows Vista 和 Windows Server 2008 上返回。
ERROR_INVALID_PARAMETER	参数不正确。如果 <i>Row</i> 参数为 NULL 指针，则返回此错误。
ERROR_INVALID_USER_BUFFER	提供给请求操作的用户缓冲区无效。如果 <i>Rw</i> 、 <i>Ros</i> 或 <i>Rod</i> 参数指向的给定缓冲区之一为 NULL ，但在关联的 <i>RwSize</i> 、 <i>RosSize</i> 或 <i>RodSize</i> 中指定了长度，则返回此错误。因此，如果满足以下任一条件，则返回此错误：

- *Row* 参数为 NULL 指针, *RwSize* 参数为非零值。
- *Ros* 参数为 NULL 指针, *RosSize* 参数为非零值。
- *Rod* 参数为 NULL 指针, *RodSize* 参数为非零值。

此错误值在 Windows 7 和 Windows Server 2008 R2 上返回。

ERROR_NOT_FOUND	找不到此请求的条目。如果找不到 <i>Row</i> 参数中指定的 TCP 连接，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果 <i>RwVersion</i> 、 <i>RosVersion</i> 或 <i>RodVersion</i> 参数未设置为零，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetPerTcpConnectionEStats](#) 函数在 Windows Vista 及更高版本上定义。

[GetPerTcpConnectionEStats](#) 函数旨在使用 TCP 诊断网络和应用程序中的性能问题。如果基于网络的应用程序性能不佳，TCP 可以确定瓶颈是在发送方、接收方还是网络本身。如果瓶颈在网络中，TCP 可以提供有关其性质的特定信息。

[GetPerTcpConnectionEStats](#) 函数检索 *Row* 参数中传递的 IPv4 TCP 连接的扩展统计信息。检索的扩展统计信息的类型在 *EstatsType* 参数中指定。以前必须通过调用 [SetPerTcpConnectionEStats](#) 函数为所有 **TCP_ESTATS_TYPE** 值启用此 TCP 连接的扩展统计信息，除非在 *EstatsType* 参数中传递 [TcpConnectionEstatsSynOpts](#)。

[GetTcpTable](#) 函数用于检索本地计算机上的 IPv4 TCP 连接表。此函数返回包含 **MIB_TCPROW** 项数组的 **MIB_TCPTABLE** 结构。传递给 [GetPerTcpConnectionEStats](#) 函数的 *Row* 参数必须是现有 IPv4 TCP 连接的条目。

当前支持的唯一 TCP 连接统计信息版本为版本零。因此，传递给 [GetPerTcpConnectionEStats](#) 的 *RwVersion*、*RosVersion* 和 *RodVersion* 参数应设置为 0。

有关 IPv6 连接的扩展 TCP 统计信息的信息，请参阅 [GetPerTcp6ConnectionEStats](#) 和 [SetPerTcp6ConnectionEStats](#) 函数。

[SetPerTcpConnectionEStats](#) 函数只能由以 Administrators 组成员身份登录的用户调用。如果 [SetPerTcpConnectionEStats](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 **ERROR_ACCESS_DENIED**。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 **requestedExecutionLevel** 设置为 **requireAdministrator**。如果应用程序缺少此清单

文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

`GetPerTcpConnectionEStats` 的调用方应在返回的 `Rw` 结构中检查 `EnableCollection` 字段，如果不是 `TRUE`，则调用方应忽略 `Ros` 和 `Rod` 结构中的数据。如果 `EnableCollection` 设置为 `FALSE`，则 `Ros` 和 `Rod` 中返回的数据未定义。例如，发生这种情况的一个条件是，使用 `GetPerTcpConnectionEStats` 检索 IPv4 TCP 连接的扩展统计信息，并且之前调用 `SetPerTcpConnectionEStats` 来启用扩展统计信息。如果 `SetPerTcpConnectionEStats` 调用失败，则对 `GetPerTcpConnectionEStats` 的后续调用将返回无意义的随机数据，而不是扩展的 TCP 统计信息。可以通过以管理员和普通用户身份运行以下示例来观察该示例。

示例

有关代码示例，请参阅 [GetPerTcp6ConnectionEStats 函数](#) 主题中的示例部分。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	<code>iphlpapi.h</code>
Library	<code>iphlpapi.lib</code>
DLL	<code>iphlpapi.dll</code>

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetTcpTable](#)

[MIB_TCPROW](#)

[MIB_TCPTABLE](#)

[SetPerTcp6ConnectionEStats](#)

SetPerTcpConnectionEStats

TCP_ESTATS_BANDWIDTH_ROD_v0

TCP_ESTATS_BANDWIDTH_RW_v0

TCP_ESTATS_DATA_ROD_v0

TCP_ESTATS_DATA_RW_v0

TCP_ESTATS_FINE_RTT_ROD_v0

TCP_ESTATS_FINE_RTT_RW_v0

TCP_ESTATS_OBS_REC_ROD_v0

TCP_ESTATS_OBS_REC_RW_v0

TCP_ESTATS_PATH_ROD_v0

TCP_ESTATS_PATH_RW_v0

TCP_ESTATS_REC_ROD_v0

TCP_ESTATS_REC_RW_v0

TCP_ESTATS_SEND_BUFF_ROD_v0

TCP_ESTATS_SEND_BUFF_RW_v0

TCP_ESTATS SND CONG ROD v0

TCP_ESTATS SND CONG ROS v0

TCP_ESTATS SND CONG RW v0

TCP_ESTATS_SYN_OPTS_ROS_v0

TCP_ESTATS_TYPE

TCP_SOFT_ERROR

反馈

此页面是否有帮助?

 是

 否

在 Microsoft Q&A 获取帮助

getRTTAndHopCount 函数 (iphlpapi.h)

项目2023/08/24

GetRTTAndHopCount 函数确定 rtT 和跃点计数 (指定目标的往返时间)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE BOOL GetRTTAndHopCount(
    [in]  IPAddr DestIpAddress,
    [out] PULONG HopCount,
    [in]  ULONG  MaxHops,
    [out] PULONG RTT
);
```

parameters

[in] DestIpAddress

要确定其 RTT 和跃点计数的目标的 IP 地址，采用 [IPAddr](#) 结构的形式。

[out] HopCount

指向 [ULONG](#) 变量的指针。此变量接收 *DestIpAddress* 参数指定的目标的跃点计数。

[in] MaxHops

要搜索目标的最大跃点数。如果目标跃点数超过此数字，函数将终止搜索并返回 FALSE。

[out] RTT

到 *DestIpAddress* 指定的目标的往返时间 (以毫秒为单位)。

返回值

如果函数成功，则返回值为 TRUE。

如果函数失败，则返回值为 FALSE。调用 [GetLastError](#) 以获取失败的错误代码。

注解

有关 **IPAddr** 数据类型的信息，请参阅 [Windows 数据类型](#)。若要在点点十进制表示法和 **IPAddr** 格式之间转换 IP 地址，请使用 [inet_addr](#) 和 [inet_ntoa](#) 函数。

示例

以下示例检索并打印目标 IP 地址 127.0.0.1 的往返时间和跃点计数。

C++

```
UINT ip = inet_addr("127.0.0.1");
ULONG hopCount = 0;
ULONG RTT = 0;

if(GetRTTAndHopCount(ip, &hopCount, 30, &RTT) == TRUE) {
    printf("Hops: %ld\n", hopCount);
    printf("RTT: %ld\n", RTT);
}
else {
    printf("Error: %ld\n", GetLastError());
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetBestInterface](#)

[GetBestRoute](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IPAddr](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getTcpStatistics 函数 (iphlpapi.h)

项目2023/08/24

GetTcpStatistics 函数检索本地计算机的 TCP 统计信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetTcpStatistics(
    [out] PMIB_TCPSTATS Statistics
);
```

parameters

[out] Statistics

指向接收本地计算机的 TCP 统计信息 的 MIB_TCPSTATS 结构的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	pStats 参数为 NULL，或者 GetTcpStatistics 无法写入 pStats 参数指向的内存。
其他	使用 FormatMessage 函数获取返回错误的消息字符串。

注解

GetTcpStatistics 函数返回当前计算机上 IPv4 的 TCP 统计信息。在 Windows XP 及更高版本上，GetTcpStatisticsEx 可用于获取 IPv4 或 IPv6 的 TCP 统计信息。

示例

以下示例检索本地计算机的 TCP 统计信息，并从返回的数据中输出一些值。

C++

```
//#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int main()
{
    PMIB_TCPSTATS pTCPStats;
    DWORD dwRetVal = 0;

    pTCPStats = (MIB_TCPSTATS*) MALLOC (sizeof(MIB_TCPSTATS));
    if (pTCPStats == NULL) {
        printf("Error allocating memory\n");
        return 1;
    }

    if ((dwRetVal = GetTcpStatistics(pTCPStats)) == NO_ERROR) {
        printf("\tActive Opens: %ld\n", pTCPStats->dwActiveOpens);
        printf("\tPassive Opens: %ld\n", pTCPStats->dwPassiveOpens);
        printf("\tSegments Recv: %ld\n", pTCPStats->dwInSegs);
        printf("\tSegments Xmit: %ld\n", pTCPStats->dwOutSegs);
        printf("\tTotal # Conxs: %ld\n", pTCPStats->dwNumConns);
    }
    else {
        printf("GetTcpStatistics failed with error: %ld\n", dwRetVal);

        LPVOID lpMsgBuf;
        if (FormatMessage( FORMAT_MESSAGE_ALLOCATE_BUFFER |
                           FORMAT_MESSAGE_FROM_SYSTEM |
                           FORMAT_MESSAGE_IGNORE_INSERTS,
                           NULL,
                           dwRetVal,
                           MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
                           (LPTSTR) &lpMsgBuf,
                           0,
                           NULL )) {
            printf("\tError: %s", lpMsgBuf);
        }
        LocalFree( lpMsgBuf );
    }

    if (pTCPStats)
        FREE (pTCPStats);
```

}

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIcmpStatistics](#)

[GetIpStatistics](#)

[GetTcpStatisticsEx](#)

[GetUdpStatistics](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_TCPSTATS](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

GetTcpStatisticsEx 函数 (iphlpapi.h)

项目2023/08/24

GetTcpStatisticsEx 函数检索当前计算机的传输控制协议 (TCP) 统计信息。

GetTcpStatisticsEx 函数与 GetTcpStatistics 函数的不同之处在于，GetTcpStatisticsEx 还支持 Internet 协议版本 6 (IPv6) 协议系列。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetTcpStatisticsEx(
    [out] PMIB_TCPSTATS Statistics,
    [in]   ULONG          Family
);
```

parameters

[out] Statistics

指向接收本地计算机的 TCP 统计信息 的 MIB_TCPSTATS 结构的指针。

[in] Family

要检索其统计信息的协议系列。此参数必须是以下值之一：

值	含义
AF_INET	Internet 协议版本 4 (IPv4)。
AF_INET6	Internet 协议版本 6 (IPv6)。

返回值

如果函数成功，则返回值 NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	pStats 参数为 NULL 或未指向有效内存，或者 dwFamily 参数不是有效值。

ERROR_NOT_SUPPORTED	执行函数调用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

要求

最低受支持的客户端	Windows XP [桌面应用 UWP 应用]
最低受支持的服务器	Windows 2000 Server [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIpStatisticsEx](#)

[GetTcpStatistics](#)

[GetUdpStatisticsEx](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_TCPSTATS](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getTcpStatisticsEx2 函数 (iphlpapi.h)

项目2023/08/24

GetTcpStatisticsEx2 函数检索当前计算机的传输控制协议 (TCP) 统计信息。

GetTcpStatisticsEx2 函数与 [GetTcpStatisticsEx](#) 函数的不同之处在于，它使用包含 64 位计数器而不是 32 位计数器的新输出结构。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetTcpStatisticsEx2(
    [out] PMIB_TCPSTATS2 Statistics,
    [in]   ULONG           Family
);
```

parameters

[out] Statistics

指向接收本地计算机的 TCP 统计信息 的 [MIB_TCPSTATS2](#) 结构的指针。

[in] Family

要检索其统计信息的协议系列。此参数必须是以下值之一：

值	含义
AF_INET	internet 协议版本 4 (IPv4)。
AF_INET6	Internet 协议版本 6 (IPv6)。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	pStats 参数为 NULL 或未指向有效内存，或者 dwFamily 参数不是有效值。

ERROR_NOT_SUPPORTED	进行函数调用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

要求

最低受支持的客户端	Windows 10 版本 1709 [仅限桌面应用]
最低受支持的服务器	Windows Server 2016 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getTcp6Table 函数 (iphlpapi.h)

项目2023/08/24

GetTcp6Table 函数检索 IPv6 的 TCP 连接表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetTcp6Table(
    [out]     PMIB_TCP6TABLE TcpTable,
    [in, out] PULONG      SizePointer,
    [in]      BOOL       Order
);
```

parameters

[out] TcpTable

指向缓冲区的指针，该缓冲区接收 IPv6 的 TCP 连接表作为 MIB_TCP6TABLE 结构。

[in, out] SizePointer

输入时，指定 *TcpTable* 参数指向的缓冲区的大小（以字节为单位）。

在输出时，如果缓冲区不够大，无法容纳返回的 TCP 连接表，则函数会将此参数设置为等于所需的缓冲区大小（以字节为单位）。

[in] Order

一个布尔值，指定是否应对 TCP 连接表进行排序。如果此参数为 TRUE，则表按升序排序，从最低本地 IP 地址开始。如果此参数为 FALSE，则表将按检索顺序显示。

在对 TCP 终结点进行排序时，将 () 列出的 (比较以下值：

1. 本地 IPv6 地址
2. 本地范围 ID
3. 本地端口
4. 远程 IPv6 地址
5. 远程范围 ID
6. 远程端口

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	<i>TcpTable</i> 参数指向的缓冲区不够大。所需的大小在 <i>SizePointer</i> 参数指向的变量中返回。
ERROR_INVALID_PARAMETER	<i>SizePointer</i> 参数为 NULL，或者 GetTcp6Table 无法写入 <i>SizePointer</i> 参数指向的内存。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetTcp6Table](#) 函数在 Windows Vista 及更高版本上定义。

示例

以下示例检索 IPv6 的 TCP 连接表，并输出每个连接的状态。

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

// Need to link with Iphlpapi.lib and Ws2_32.lib
#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))
/* Note: could also use malloc() and free() */

int wmain()
{
    // Declare and initialize variables
```

```

PMIB_TCP6TABLE pTcpTable;
DWORD dwSize = 0;
DWORD dwRetVal = 0;

wchar_t ipstringbuffer[46];

int i;

pTcpTable = (MIB_TCP6TABLE *) MALLOC(sizeof(MIB_TCP6TABLE));
if (pTcpTable == NULL) {
    wprintf(L"Error allocating memory\n");
    return 1;
}

dwSize = sizeof(MIB_TCP6TABLE);
// Make an initial call to GetTcp6Table to
// get the necessary size into the dwSize variable
if ((dwRetVal = GetTcp6Table(pTcpTable, &dwSize, TRUE)) ==
    ERROR_INSUFFICIENT_BUFFER) {
    FREE(pTcpTable);
    pTcpTable = (MIB_TCP6TABLE *) MALLOC(dwSize);
    if (pTcpTable == NULL) {
        wprintf(L"Error allocating memory\n");
        return 1;
    }
}
// Make a second call to GetTcp6Table to get
// the actual data we require
if ((dwRetVal = GetTcp6Table(pTcpTable, &dwSize, TRUE)) == NO_ERROR) {
    wprintf(L"\tNumber of entries: %d\n", (int) pTcpTable-
>dwNumEntries);
    for (i = 0; i < (int) pTcpTable->dwNumEntries; i++) {
        wprintf(L"\n\tTCP[%d] State: %ld - ", i,
               pTcpTable->table[i].State);
        switch (pTcpTable->table[i].State) {
        case MIB_TCP_STATE_CLOSED:
            wprintf(L"CLOSED\n");
            break;
        case MIB_TCP_STATE_LISTEN:
            wprintf(L"LISTEN\n");
            break;
        case MIB_TCP_STATE_SYN_SENT:
            wprintf(L"SYN-SENT\n");
            break;
        case MIB_TCP_STATE_SYN_RCVD:
            wprintf(L"SYN-RECEIVED\n");
            break;
        case MIB_TCP_STATE_ESTAB:
            wprintf(L"ESTABLISHED\n");
            break;
        case MIB_TCP_STATE_FIN_WAIT1:
            wprintf(L"FIN-WAIT-1\n");
            break;
        case MIB_TCP_STATE_FIN_WAIT2:
            wprintf(L"FIN-WAIT-2 \n");
        }
    }
}

```

```

        break;
    case MIB_TCP_STATE_CLOSE_WAIT:
        wprintf(L"CLOSE-WAIT\n");
        break;
    case MIB_TCP_STATE_CLOSING:
        wprintf(L"CLOSING\n");
        break;
    case MIB_TCP_STATE_LAST_ACK:
        wprintf(L"LAST-ACK\n");
        break;
    case MIB_TCP_STATE_TIME_WAIT:
        wprintf(L"TIME-WAIT\n");
        break;
    case MIB_TCP_STATE_DELETE_TCB:
        wprintf(L"DELETE-TCB\n");
        break;
    default:
        wprintf(L"UNKNOWN dwState value\n");
        break;
    }

    if (InetNtop(AF_INET6, &pTcpTable->table[i].LocalAddr,
ipstringbuffer, 46) == NULL)
        wprintf(L"  InetNtop function failed for local IPv6
address\n");
    else
        wprintf(L"\tTCP[%d] Local Addr: %s\n", i, ipstringbuffer);
        wprintf(L"\tTCP[%d] Local Scope ID: %d \n", i,
            ntohl (pTcpTable->table[i].dwLocalScopeId));
        wprintf(L"\tTCP[%d] Local Port: %d \n", i,
            ntohs((u_short)pTcpTable->table[i].dwLocalPort));

        if (InetNtop(AF_INET6, &pTcpTable->table[i].RemoteAddr,
ipstringbuffer, 46) == NULL)
            wprintf(L"  InetNtop function failed for remote IPv6
address\n");
        else
            wprintf(L"\tTCP[%d] Remote Addr: %s\n", i, ipstringbuffer);
            wprintf(L"\tTCP[%d] Remote Scope ID: %d \n", i,
                ntohl(pTcpTable->table[i].dwRemoteScopeId));
            wprintf(L"\tTCP[%d] Remote Port: %d\n", i,
                ntohs((u_short)pTcpTable->table[i].dwRemotePort));
    }
} else {
    wprintf(L"\tGetTcp6Table failed with %d\n", dwRetVal);
    FREE(pTcpTable);
    return 1;
}

if (pTcpTable != NULL) {
    FREE(pTcpTable);
    pTcpTable = NULL;
}

return 0;

```

}

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetExtendedTcpTable](#)

[GetOwnerModuleFromTcp6Entry](#)

[GetTcp6Table2](#)

[GetTcpStatisticsEx](#)

[GetTcpTable2](#)

[MIB_TCP6ROW](#)

[MIB_TCP6ROW_OWNER_MODULE](#)

[MIB_TCP6ROW_OWNER_PID](#)

[MIB_TCP6TABLE](#)

[MIB_TCP6TABLE_OWNER_MODULE](#)

[MIB_TCP6TABLE_OWNER_PID](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

getTcp6Table2 函数 (iphlpapi.h)

项目2023/08/24

GetTcp6Table2 函数检索 IPv6 的 TCP 连接表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetTcp6Table2(
    [out]     PMIB_TCP6TABLE2 TcpTable,
    [in, out] PULONG          SizePointer,
    [in]      BOOL           Order
);
```

parameters

[out] *TcpTable*

指向缓冲区的指针，该缓冲区接收 IPv6 的 TCP 连接表作为 MIB_TCP6TABLE2 结构。

[in, out] *SizePointer*

输入时，指定 *TcpTable* 参数指向的缓冲区的大小。

在输出中，如果缓冲区不够大，无法容纳返回的 TCP 连接表，则函数会将此参数设置为所需的缓冲区大小。

[in] *Order*

一个值，该值指定是否应对 TCP 连接表进行排序。如果此参数为 TRUE，则表按升序排序，从最低本地 IP 地址开始。如果此参数为 FALSE，则表将按检索顺序显示。

在对 TCP 终结点进行排序时，将 () 列出的 (比较以下值：

1. 本地 IPv6 地址
2. 本地范围 ID
3. 本地端口
4. 远程 IPv6 地址
5. 远程范围 ID
6. 远程端口

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	<i>TcpTable</i> 参数指向的缓冲区不够大。所需的大小在 <i>SizePointer</i> 参数指向的变量中返回。
ERROR_INVALID_PARAMETER	<i>SizePointer</i> 参数为 NULL，或者 GetTcp6Table2 无法写入 <i>SizePointer</i> 参数指向的内存。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetTcp6Table2](#) 函数在 Windows Vista 及更高版本上定义。

[GetTcp6Table2](#) 函数是 [GetTcp6Table](#) 函数的增强版本，它还检索有关 TCP 连接的 TCP 卸载状态的信息。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[GetExtendedTcpTable](#)

[GetOwnerModuleFromTcp6Entry](#)

[GetTcp6Table](#)

[GetTcpStatisticsEx](#)

[GetTcpTable](#)

[MIB_TCP6ROW](#)

[MIB_TCP6ROW2](#)

[MIB_TCP6ROW_OWNER_MODULE](#)

[MIB_TCP6ROW_OWNER_PID](#)

[MIB_TCP6TABLE](#)

[MIB_TCP6TABLE2](#)

[MIB_TCP6TABLE_OWNER_MODULE](#)

[MIB_TCP6TABLE_OWNER_PID](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

getTcpTable 函数 (iphlpapi.h)

项目2023/08/24

GetTcpTable 函数检索 IPv4 TCP 连接表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetTcpTable(
    [out]      PMIB_TCPTABLE TcpTable,
    [in, out]   PULONG      SizePointer,
    [in]       BOOL        Order
);
```

parameters

[out] TcpTable

指向以MIB_TCPTABLE结构的形式接收 TCP 连接表 [的缓冲区的](#) 指针。

[in, out] SizePointer

在输入时，指定 *pTcpTable* 参数指向的缓冲区的大小（以字节为单位）。

在输出时，如果缓冲区不够大，无法容纳返回的连接表，则函数会将此参数设置为等于所需的缓冲区大小（以字节为单位）。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，此参数的数据类型将更改为等效于 PDWORD 的 PULONG。

[in] Order

一个布尔值，指定是否应对 TCP 连接表进行排序。如果此参数为 TRUE，则按以下顺序对表进行排序：

1. 本地 IP 地址
2. 本地端口
3. 远程 IP 地址
4. 远程端口

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	<i>pTcpTable</i> 参数指向的缓冲区不够大。所需大小在 <i>pdwSize</i> 参数指向的 DWORD 变量中返回。 如果 <i>pTcpTable</i> 参数为 NULL，也会返回此错误。
ERROR_INVALID_PARAMETER	<i>pdwSize</i> 参数为 NULL，或者 GetTcpTable 无法写入 <i>pdwSize</i> 参数指向的内存。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
STATUS_UNSUCCESSFUL	如果收到此返回代码，则再次调用函数通常足以清除问题并获得所需的结果。此返回代码可能是系统负载过高的结果。例如，如果 TCP 连接表的大小连续 3 次更改超过 2 个其他项。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

在为 Windows Vista 及更高版本发布的 Windows SDK 上，[GetTcpTable](#) 函数的返回值将更改为等效于 DWORD 的 ULONG 数据类型。

示例

以下示例检索 IPv4 的 TCP 连接表，并输出每个连接的状态。

C++

```
// Need to link with Iphlpapi.lib and Ws2_32.lib
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int main()
```

```

{

// Declare and initialize variables
PMIB_TCPTABLE pTcpTable;
DWORD dwSize = 0;
DWORD dwRetVal = 0;

char szLocalAddr[128];
char szRemoteAddr[128];

struct in_addr IpAddr;

int i;

pTcpTable = (MIB_TCPTABLE *) MALLOC(sizeof(MIB_TCPTABLE));
if (pTcpTable == NULL) {
    printf("Error allocating memory\n");
    return 1;
}

dwSize = sizeof(MIB_TCPTABLE);
// Make an initial call to GetTcpTable to
// get the necessary size into the dwSize variable
if ((dwRetVal = GetTcpTable(pTcpTable, &dwSize, TRUE)) ==
    ERROR_INSUFFICIENT_BUFFER) {
    FREE(pTcpTable);
    pTcpTable = (MIB_TCPTABLE *) MALLOC(dwSize);
    if (pTcpTable == NULL) {
        printf("Error allocating memory\n");
        return 1;
    }
}
// Make a second call to GetTcpTable to get
// the actual data we require
if ((dwRetVal = GetTcpTable(pTcpTable, &dwSize, TRUE)) == NO_ERROR) {
    printf("\tNumber of entries: %d\n", (int) pTcpTable->dwNumEntries);
    for (i = 0; i < (int) pTcpTable->dwNumEntries; i++) {
        IpAddr.S_un.S_addr = (u_long) pTcpTable->table[i].dwLocalAddr;
        strcpy_s(szLocalAddr, sizeof(szLocalAddr), inet_ntoa(IpAddr));
        IpAddr.S_un.S_addr = (u_long) pTcpTable->table[i].dwRemoteAddr;
        strcpy_s(szRemoteAddr, sizeof(szRemoteAddr),
inet_ntoa(IpAddr));

        printf("\n\tTCP[%d] State: %ld - ", i,
               pTcpTable->table[i].dwState);
        switch (pTcpTable->table[i].dwState) {
        case MIB_TCP_STATE_CLOSED:
            printf("CLOSED\n");
            break;
        case MIB_TCP_STATE_LISTEN:
            printf("LISTEN\n");
            break;
        case MIB_TCP_STATE_SYN_SENT:
            printf("SYN-SENT\n");
            break;
        }
    }
}

```

```

        case MIB_TCP_STATE_SYN_RCVD:
            printf("SYN-RECEIVED\n");
            break;
        case MIB_TCP_STATE_ESTAB:
            printf("ESTABLISHED\n");
            break;
        case MIB_TCP_STATE_FIN_WAIT1:
            printf("FIN-WAIT-1\n");
            break;
        case MIB_TCP_STATE_FIN_WAIT2:
            printf("FIN-WAIT-2 \n");
            break;
        case MIB_TCP_STATE_CLOSE_WAIT:
            printf("CLOSE-WAIT\n");
            break;
        case MIB_TCP_STATE_CLOSING:
            printf("CLOSING\n");
            break;
        case MIB_TCP_STATE_LAST_ACK:
            printf("LAST-ACK\n");
            break;
        case MIB_TCP_STATE_TIME_WAIT:
            printf("TIME-WAIT\n");
            break;
        case MIB_TCP_STATE_DELETE_TCB:
            printf("DELETE-TCB\n");
            break;
        default:
            printf("UNKNOWN dwState value\n");
            break;
    }
    printf("\tTCP[%d] Local Addr: %s\n", i, szLocalAddr);
    printf("\tTCP[%d] Local Port: %d \n", i,
           ntohs((u_short)pTcpTable->table[i].dwLocalPort));
    printf("\tTCP[%d] Remote Addr: %s\n", i, szRemoteAddr);
    printf("\tTCP[%d] Remote Port: %d\n", i,
           ntohs((u_short)pTcpTable->table[i].dwRemotePort));
}
} else {
    printf("\tGetTcpTable failed with %d\n", dwRetVal);
    FREE(pTcpTable);
    return 1;
}

if (pTcpTable != NULL) {
    FREE(pTcpTable);
    pTcpTable = NULL;
}

return 0;
}

```

要求

最低受支持的客户端	Windows 2000 专业版 [桌面应用 UWP 应用]
最低受支持的服务器	Windows 2000 Server [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetExtendedTcpTable](#)

[GetOwnerModuleFromTcpEntry](#)

[GetTcp6Table](#)

[GetTcp6Table2](#)

[GetTcpStatistics](#)

[GetTcpStatisticsEx](#)

[GetTcpTable2](#)

[MIB_TCPROW](#)

[MIB_TCPROW_OWNER_MODULE](#)

[MIB_TCPROW_OWNER_PID](#)

[MIB_TCPTABLE](#)

[MIB_TCPTABLE_OWNER_MODULE](#)

[MIB_TCPTABLE_OWNER_PID](#)

[SetTcpEntry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getTcpTable2 函数 (iphlpapi.h)

项目2023/08/24

GetTcpTable2 函数检索 IPv4 TCP 连接表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetTcpTable2(
    [out]      PMIB_TCPTABLE2 TcpTable,
    [in, out]   PULONG       SizePointer,
    [in]        BOOL        Order
);
```

parameters

[out] `TcpTable`

指向缓冲区的指针，该缓冲区接收 TCP 连接表作为 `MIB_TCPTABLE2` 结构。

[in, out] `SizePointer`

在输入时，指定 `TcpTable` 参数指向的缓冲区的大小。

在输出时，如果缓冲区不够大，无法容纳返回的连接表，则函数会将此参数设置为所需的缓冲区大小。

[in] `Order`

一个值，该值指定是否应对 TCP 连接表进行排序。如果此参数为 TRUE，则按以下顺序对表进行排序：

1. 本地 IP 地址
2. 本地端口
3. 远程 IP 地址
4. 远程端口

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	<i>TcpTable</i> 参数指向的缓冲区不够大。所需大小在 <i>SizePointer</i> 参数指向的 PULONG 变量中返回。 如果 <i>pTcpTable</i> 参数为 NULL ，也会返回此错误。
ERROR_INVALID_PARAMETER	<i>SizePointer</i> 参数为 NULL ，或者 GetTcpTable2 无法写入 <i>SizePointer</i> 参数指向的内存。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetTcpTable2](#) 函数在 Windows Vista 及更高版本上定义。

[GetTcpTable2](#) 函数是 [GetTcpTable](#) 函数的增强版本，它还检索有关 TCP 连接的 TCP 卸载状态的信息。

示例

以下示例检索 IPv4 的 TCP 连接表，并输出每个连接的状态。

C++

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

// Need to link with Iphlpapi.lib and Ws2_32.lib
#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))
/* Note: could also use malloc() and free() */

int main()
{
    // Declare and initialize variables
    PMIB_TCPTABLE2 pTcpTable;
    ULONG ulSize = 0;
    DWORD dwRetVal = 0;
```

```

char szLocalAddr[128];
char szRemoteAddr[128];

struct in_addr IpAddr;

int i;

pTcpTable = (MIB_TCPTABLE2 *) MALLOC(sizeof (MIB_TCPTABLE2));
if (pTcpTable == NULL) {
    printf("Error allocating memory\n");
    return 1;
}

ulSize = sizeof (MIB_TCPTABLE);
// Make an initial call to GetTcpTable2 to
// get the necessary size into the ulSize variable
if ((dwRetVal = GetTcpTable2(pTcpTable, &ulSize, TRUE)) ==
    ERROR_INSUFFICIENT_BUFFER) {
    FREE(pTcpTable);
    pTcpTable = (MIB_TCPTABLE2 *) MALLOC(ulSize);
    if (pTcpTable == NULL) {
        printf("Error allocating memory\n");
        return 1;
    }
}
// Make a second call to GetTcpTable2 to get
// the actual data we require
if ((dwRetVal = GetTcpTable2(pTcpTable, &ulSize, TRUE)) == NO_ERROR) {
    printf("\tNumber of entries: %d\n", (int) pTcpTable->dwNumEntries);
    for (i = 0; i < (int) pTcpTable->dwNumEntries; i++) {
        printf("\n\tTCP[%d] State: %ld - ", i,
               pTcpTable->table[i].dwState);
        switch (pTcpTable->table[i].dwState) {
        case MIB_TCP_STATE_CLOSED:
            printf("CLOSED\n");
            break;
        case MIB_TCP_STATE_LISTEN:
            printf("LISTEN\n");
            break;
        case MIB_TCP_STATE_SYN_SENT:
            printf("SYN-SENT\n");
            break;
        case MIB_TCP_STATE_SYN_RCVD:
            printf("SYN-RECEIVED\n");
            break;
        case MIB_TCP_STATE_ESTAB:
            printf("ESTABLISHED\n");
            break;
        case MIB_TCP_STATE_FIN_WAIT1:
            printf("FIN-WAIT-1\n");
            break;
        case MIB_TCP_STATE_FIN_WAIT2:
            printf("FIN-WAIT-2 \n");
            break;
        }
    }
}

```

```

    case MIB_TCP_STATE_CLOSE_WAIT:
        printf("CLOSE-WAIT\n");
        break;
    case MIB_TCP_STATE_CLOSING:
        printf("CLOSING\n");
        break;
    case MIB_TCP_STATE_LAST_ACK:
        printf("LAST-ACK\n");
        break;
    case MIB_TCP_STATE_TIME_WAIT:
        printf("TIME-WAIT\n");
        break;
    case MIB_TCP_STATE_DELETE_TCB:
        printf("DELETE-TCB\n");
        break;
    default:
        printf("UNKNOWN dwState value\n");
        break;
    }
    IpAddr.S_un.S_addr = (u_long) pTcpTable->table[i].dwLocalAddr;
    strcpy_s(szLocalAddr, sizeof (szLocalAddr), inet_ntoa(IpAddr));
    printf("\tTCP[%d] Local Addr: %s\n", i, szLocalAddr);
    printf("\tTCP[%d] Local Port: %d \n", i,
           ntohs((u_short)pTcpTable->table[i].dwLocalPort));

    IpAddr.S_un.S_addr = (u_long) pTcpTable->table[i].dwRemoteAddr;
    strcpy_s(szRemoteAddr, sizeof (szRemoteAddr),
            inet_ntoa(IpAddr));
    printf("\tTCP[%d] Remote Addr: %s\n", i, szRemoteAddr);
    printf("\tTCP[%d] Remote Port: %d\n", i,
           ntohs((u_short)pTcpTable->table[i].dwRemotePort));

    printf("\tTCP[%d] Owning PID: %d\n", i, pTcpTable-
>table[i].dwOwningPid);
    printf("\tTCP[%d] Offload State: %ld - ", i,
           pTcpTable->table[i].dwOffloadState);
    switch (pTcpTable->table[i].dwOffloadState) {
    case TcpConnectionOffloadStateInHost:
        printf("Owned by the network stack and not offloaded \n");
        break;
    case TcpConnectionOffloadStateOffloading:
        printf("In the process of being offloaded\n");
        break;
    case TcpConnectionOffloadStateOffloaded:
        printf("Offloaded to the network interface control\n");
        break;
    case TcpConnectionOffloadStateUploading:
        printf("In the process of being uploaded back to the network
stack \n");
        break;
    default:
        printf("UNKNOWN Offload state value\n");
        break;
    }

```

```

    }
} else {
    printf("\tGetTcpTable2 failed with %d\n", dwRetVal);
    FREE(pTcpTable);
    return 1;
}

if (pTcpTable != NULL) {
    FREE(pTcpTable);
    pTcpTable = NULL;
}

return 0;
}

```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[GetExtendedTcpTable](#)

[GetOwnerModuleFromTcpEntry](#)

[GetTcp6Table](#)

[GetTcpStatistics](#)

[GetTcpStatisticsEx](#)

[GetTcpTable](#)

[MIB_TCP6ROW2](#)

[MIB_TCP6TABLE](#)

[MIB_TCP6TABLE2](#)

[MIB_TCP6TABLE_OWNER_MODULE](#)

[MIB_TCP6TABLE_OWNER_PID](#)

[MIB_TCPROW](#)

[MIB_TCPROW2](#)

[MIB_TCPROW_OWNER_MODULE](#)

[MIB_TCPROW_OWNER_PID](#)

[MIB_TCPTABLE](#)

[MIB_TCPTABLE2](#)

[MIB_TCPTABLE_OWNER_MODULE](#)

[MIB_TCPTABLE_OWNER_PID](#)

[SetTcpEntry](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

getTeredoPort 函数 (netioapi.h)

项目2023/08/25

GetTeredoPort 函数检索本地计算机上 Teredo 客户端使用的动态 UDP 端口号。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API GetTeredoPort(
    [out] USHORT *Port
);
```

parameters

[out] Port

指向 UDP 端口号的指针。 成功返回后，此参数将填充 Teredo 客户端使用的端口号。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果在 <i>Port</i> 参数中传递 NULL 指针，则返回此错误。
ERROR_NOT_READY	设备未准备就绪。 如果未在本地计算机上启动 Teredo 客户端，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。 如果本地计算机上没有 IPv6 堆栈，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

GetTeredoPort 函数在 Windows Vista 及更高版本上定义。

GetTeredoPort 函数检索 Teredo 客户端用于 Teredo 服务端口的当前 UDP 端口号。

Teredo 端口是动态的，可以在本地计算机上重启 Teredo 客户端时随时更改。应用程序可以通过调用 [NotifyTeredoPortChange](#) 函数注册，以便在 Teredo 服务端口更改时收到通知。

Teredo 客户端还使用静态 UDP 端口 3544 来侦听在 RFC 4380 中定义的多播 IPv4 地址 224.0.0.253 上发送的多播流量。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4380.txt>。

GetTeredoPort 函数主要由防火墙应用程序使用，以便配置相应的例外以允许传入和传出 Teredo 流量。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[NotifyStableUnicastIpAddressTable](#)

[NotifyTeredoPortChange](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

getUpdStatistics 函数 (iphlpapi.h)

项目2023/08/24

GetUpdStatistics 函数检索本地计算机的用户数据报协议 (UDP) 统计信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetUpdStatistics(
    [out] PMIB_UDPSTATS Stats
);
```

parameters

[out] Stats

指向接收本地计算机的 UDP 统计信息 的MIB_UDPSTATS 结构的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，请使用 FormatMessage 获取返回错误的消息字符串。

注解

Windows Server 2003 和 Windows XP： 使用 GetUpdStatisticsEx 函数获取 IPv6 协议的 UDP 统计信息。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows

标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIcmpStatistics](#)

[GetIpStatistics](#)

[GetTcpStatistics](#)

[GetUdpStatisticsEx](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_UDPSTATS](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getUdpStatisticsEx 函数 (iphlpapi.h)

项目2023/08/24

GetUdpStatisticsEx 函数检索当前计算机的用户数据报协议 (UDP) 统计信息。GetUdpStatisticsEx 函数与 GetUdpStatistics 函数的不同之处在于，GetUdpStatisticsEx 还支持 Internet 协议版本 6 (IPv6) 协议系列。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetUdpStatisticsEx(
    [out] PMIB_UDPSTATS Statistics,
    [in]   ULONG           Family
);
```

parameters

[out] Statistics

指向接收本地计算机的 UDP 统计信息 的 MIB_UDPSTATS 结构的指针。

[in] Family

要检索其统计信息的协议系列。此参数必须是以下值之一：

值	含义
AF_INET	internet 协议版本 4 (IPv4)。
AF_INET6	Internet 协议版本 6 (IPv6)。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	pStats 参数为 NULL 或未指向有效内存，或者 dwFamily 参数不是有效值。

ERROR_NOT_SUPPORTED	进行函数调用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

要求

最低受支持的客户端	Windows XP [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2003 [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIpStatisticsEx](#)

[GetTcpStatisticsEx](#)

[GetUdpStatistics](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_UDPSTATS](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getUdpStatisticsEx2 函数 (iphlpapi.h)

项目2023/08/24

GetUdpStatisticsEx2 函数检索当前计算机的用户数据报协议 (UDP) 统计信息。

GetUdpStatisticsEx2 函数与 [GetUdpStatisticsEx](#) 函数的不同之处在于，

GetUdpStatisticsEx2 使用包含 64 位计数器而不是 32 位计数器的新输出结构。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetUdpStatisticsEx2(
    [out] PMIB_UDPSTATS2 Statistics,
    [in]   ULONG           Family
);
```

parameters

[out] Statistics

指向接收本地计算机的 UDP 统计信息 的 MIB_UDPSTATS2 结构的指针。

[in] Family

要检索其统计信息的协议系列。 此参数必须是以下值之一：

值	含义
AF_INET	internet 协议版本 4 (IPv4)。
AF_INET6	Internet 协议版本 6 (IPv6)。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	Statistics 参数为 NULL 或未指向有效内存，或者 Family 参数不是有效值。

ERROR_NOT_SUPPORTED	进行函数调用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

要求

最低受支持的客户端	Windows 10 版本 1709 [仅限桌面应用]
最低受支持的服务器	Windows Server 2016 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[GetIpStatisticsEx](#)

[GetTcpStatisticsEx](#)

[GetUdpStatistics](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_UDPSTATS](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getUdpTable 函数 (iphlpapi.h)

项目2023/08/24

GetUdpTable 函数 (UDP) 倾听器表检索 IPv4 用户数据报协议。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetUdpTable(
    [out]     PMIB_UDPTABLE UdpTable,
    [in, out] PULONG       SizePointer,
    [in]      BOOL        Order
);
```

parameters

[out] UdpTable

指向缓冲区的指针，该缓冲区接收 IPv4 UDP 倾听器表作为 MIB_UDPTABLE 结构。

[in, out] SizePointer

输入时，指定 *UdpTable* 参数指向的缓冲区的大小（以字节为单位）。

在输出时，如果缓冲区不够大，无法容纳返回的倾听器表，则函数会将此参数设置为等于所需的缓冲区大小（以字节为单位）。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，此参数的数据类型将更改为等效于 PDWORD 的 PULONG。

[in] Order

一个布尔值，指定是否应对返回的 UDP 倾听器表进行排序。如果此参数为 TRUE，则按以下顺序对表进行排序：

1. 本地 IP 地址
2. 本地端口

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	<i>pUdpTable</i> 参数指向的缓冲区不够大。所需大小在 <i>pdwSize</i> 参数指向的 ULONG 变量中返回。 如果 <i>pUdpTable</i> 参数为 NULL，也会返回此错误。
ERROR_INVALID_PARAMETER	<i>pdwSize</i> 参数为 NULL，或者 GetUdpTable 无法写入 <i>pdwSize</i> 参数指向的内存。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，[GetUdpTable](#) 函数的返回值将更改为与 DWORD 等效的 ULONG 数据类型。

要求

最低受支持的客户端	Windows 2000 专业版 [桌面应用 UWP 应用]
最低受支持的服务器	Windows 2000 Server [桌面应用 UWP 应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[GetExtendedUdpTable](#)

[GetOwnerModuleFromUdp6Entry](#)

[GetOwnerModuleFromUdpEntry](#)

[GetUdp6Table](#)

[GetUdpStatistics](#)

[GetUdpStatisticsEx](#)

[MIB_UDPROW](#)

[MIB_UDPROW_OWNER_MODULE](#)

[MIB_UDPROW_OWNER_PID](#)

[MIB_UDPTABLE](#)

[MIB_UDPTABLE_OWNER_MODULE](#)

[MIB_UDPTABLE_OWNER_PID](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

getUdp6Table 函数 (iphlpapi.h)

项目2023/08/24

GetUdp6Table 函数 (UDP) 侦听器表检索 IPv6 用户数据报协议。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG GetUdp6Table(
    [out]     PMIB_UDP6TABLE Udp6Table,
    [in, out] PULONG        SizePointer,
    [in]      BOOL          Order
);
```

parameters

[out] Udp6Table

指向以MIB_UDP6TABLE结构的形式接收 IPv6 UDP 侦听器表 的 缓冲区的指针。

[in, out] SizePointer

输入时，指定 *Udp6Table* 参数指向的缓冲区的大小（以字节为单位）。

在输出时，如果缓冲区不够大，无法容纳返回的侦听器表，则函数会将此参数设置为等于所需的缓冲区大小（以字节为单位）。

[in] Order

一个布尔值，指定是否应对返回的 UDP 侦听器表进行排序。如果此参数为 TRUE，则按以下顺序对表进行排序：

1. 本地 IPv6 地址
2. 本地范围 ID
3. 本地端口

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	<i>Udp6Table</i> 参数指向的缓冲区不够大。所需大小在 <i>SizePointer</i> 参数指向的 ULONG 变量中返回。
ERROR_INVALID_PARAMETER	<i>SizePointer</i> 参数为 NULL，或者 GetUdp6Table 无法写入 <i>SizePointer</i> 参数指向的内存。
ERROR_NOT_SUPPORTED	本地系统上正在使用的操作系统不支持此函数。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetUdp6Table](#) 函数在 Windows Vista 及更高版本上定义。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[GetExtendedUdpTable](#)

[GetOwnerModuleFromUdp6Entry](#)

[GetOwnerModuleFromUdpEntry](#)

[GetUdp6Table](#)

[GetUdpStatistics](#)

[GetUdpStatisticsEx](#)

MIB_UDP6ROW

MIB_UDP6ROW_OWNER_MODULE

MIB_UDP6ROW_OWNER_PID

MIB_UDP6TABLE

MIB_UDP6TABLE_OWNER_MODULE

MIB_UDP6TABLE_OWNER_PID

MIB_UDPROW

MIB_UDPROW_OWNER_MODULE

MIB_UDPROW_OWNER_PID

MIB_UDPTABLE

MIB_UDPTABLE_OWNER_MODULE

MIB_UDPTABLE_OWNER_PID

反馈

此页面是否有帮助?

是

否

在 Microsoft Q&A 获得帮助

getUnicastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

GetUnicastIpAddressEntry 函数检索本地计算机上现有单播 IP 地址条目的信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetUnicastIpAddressEntry(
    [in, out] PMIB_UNICASTIPADDRESS_ROW Row
);
```

parameters

[in, out] Row

指向单播 IP 地址条目 [MIB_UNICASTIPADDRESS_ROW](#) 结构条目的指针。成功返回后，将使用现有单播 IP 地址的属性更新此结构。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果由 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	参数不正确。如果在 Row 参数中传递 NULL 指针，Row 参数指向 MIB_UNICASTIPADDRESS_ROW 的 Address 成员未设置为有效的单播 IPv4 或 IPv6 地址，或者 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW 的 InterfaceLuid 和 InterfaceIndex 成员均未指定，则返回此错误。

ERROR_NOT_FOUND	找不到元素。如果 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW 结构的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口与 MIB_UNICASTIPADDRESS_ROW 结构中的 Address 成员中指定的 IP 地址不匹配，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW 结构的 Address 成员中指定了 IPv4 地址，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且 地址成员中 指定了 IPv6 地址，也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetUnicastIpAddressEntry](#) 函数在 Windows Vista 及更高版本上定义。

[GetUnicastIpAddressEntry](#) 函数通常用于检索要修改的现有 [MIB_UNICASTIPADDRESS_ROW](#) 结构条目。然后，应用程序可以更改要修改的 [MIB_UNICASTIPADDRESS_ROW](#) 条目中的成员，然后调用 [SetUnicastIpAddressEntry](#) 函数。

输入时，Row 参数指向的 [MIB_UNICASTIPADDRESS_ROW](#) 结构中的 Address 成员必须初始化为有效的单播 IPv4 或 IPv6 地址。Address 成员中 SOCKADDR_INET 结构的 si_family 成员必须初始化为 AF_INET 或 AF_INET6，并且必须将 SOCKADDR_INET 结构的相关 Ipv4 或 Ipv6 成员设置为有效的单播 IP 地址。此外，必须初始化指向 Row 参数的 [MIB_UNICASTIPADDRESS_ROW](#) 结构中的至少一个成员：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则使用此成员来确定接口。如果未为 InterfaceLuid 成员设置任何值，(此成员的值) 设置为零，则接下来使用 InterfaceIndex 成员来确定接口。

在调用成功时输出时，[GetUnicastIpAddressEntry](#) 检索单播 IP 地址的其他属性，并填写 Row 参数指向的 [MIB_UNICASTIPADDRESS_ROW](#) 结构。

可以调用 [GetUnicastIpAddressTable](#) 函数来枚举本地计算机上的单播 IP 地址条目。

示例

以下示例检索命令行中指定的单播 IP 地址条目，并从检索到的 [MIB_UNICASTIPADDRESS_ROW](#) 结构中输出一些值。

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <Windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

// Need to link with Iphlpapi.lib and Ws2_32.lib
#pragma comment (lib, "iphlpapi.lib")
#pragma comment (lib, "Ws2_32.lib")

void PrintUnicastIpAddress(PMIB_UNICASTIPADDRESS_ROW pIpRow);

int __cdecl wmain(int argc, WCHAR **argv)
{

    // Declare and initialize variables

    ULONG Result = 0;
    ULONG ifIndex;

    // default to unspecified address family
    ULONG addressFamily = AF_UNSPEC;

    IN_ADDR Ipv4Addr;
    IN6_ADDR Ipv6Addr;

    MIB_UNICASTIPADDRESS_ROW ipRow = {0};

    // Validate the parameters
    if (argc < 4) {
        wprintf(L"usage: %s <AddressFamily> <IPAddress> <InterfaceIndex>\n",
        argv[0]);
        wprintf(L"    Gets the UnicastIpAddressEntry for an
AddressFamily,\n");
        wprintf(L"    Interface Index, and IP address\n");
        wprintf(L"    Examples\n");
        wprintf(L"        Get the IPv4 loopback at interface index=1\n");
        wprintf(L"        %s 4 127.0.0.1 1\n", argv[0]);
        wprintf(L"        Get the IPv6 loopback at interface index=1\n");
        wprintf(L"        %s 6 ::1 1\n", argv[0]);
        exit(1);
    }
}
```

```

}

if (_wtoi(argv[1]) == 4) {
    addressFamily = AF_INET;
    if (InetPtonW(addressFamily, argv[2], &Ipv4Addr) != 1) {
        wprintf(L"Unable to parse IPv4 address string: %s\n", argv[3]);
        exit(1);
    }
} else if (_wtoi(argv[1]) == 6) {
    addressFamily = AF_INET6;
    if (InetPton(addressFamily, argv[2], &Ipv6Addr) != 1) {
        wprintf(L"Unable to parse IPv6 address string: %s\n", argv[3]);
        exit(1);
    }
}

ifIndex = _wtoi(argv[3]);

ipRow.Address.si_family = (ADDRESS_FAMILY) addressFamily;
ipRow.InterfaceIndex = ifIndex;

if (addressFamily == AF_INET) {
    ipRow.Address.si_family = AF_INET;
    memcpy(&ipRow.Address.Ipv4.sin_addr, &Ipv4Addr, sizeof(IN_ADDR));
}
if (addressFamily == AF_INET6) {
    ipRow.Address.si_family = AF_INET6;
    memcpy(&ipRow.Address.Ipv6.sin6_addr, &Ipv6Addr, sizeof(IN6_ADDR));
}

Result = GetUnicastIpAddressEntry(&ipRow);
if (Result != NO_ERROR) {
    wprintf(L"GetUnicastIpAddressEntry returned error: %lu\n", Result);
    exit(1);
}
PrintUnicastIpAddress(&ipRow);

exit(0);
}

void PrintUnicastIpAddress(PMIB_UNICASTIPADDRESS_ROW pipRow)
{
    WCHAR Ipv4String[16] = { 0 };
    WCHAR Ipv6String[46] = { 0 };

    // Print some variables from the rows in the table
    wprintf(L"AddressFamily:\t\t\t");
    switch (pipRow->Address.si_family) {
    case AF_INET:
        wprintf(L"IPv4\n");
        if (InetNtop(AF_INET, &pipRow->Address.Ipv4.sin_addr, Ipv4String,
16) !=
            NULL)
            wprintf(L"IPv4 Address:\t\t\t%ws\n", Ipv4String);

```

```

        break;
    case AF_INET6:
        wprintf(L"IPv6\n");
        if (InetNtop(AF_INET6, &pipRow->Address.Ipv6.sin6_addr, Ipv6String,
46) != NULL)
            wprintf(L"IPv6 Address:\t\t\t %s\n", Ipv6String);
        break;
    default:
        wprintf(L"Other: %d\n", pipRow->Address.si_family);
        break;
    }

    wprintf(L"Interface LUID NetLuidIndex:\t %lu\n",
           pipRow->InterfaceLuid.Info.NetLuidIndex);
    wprintf(L"Interface LUID IfType:\t\t ");
    switch (pipRow->InterfaceLuid.Info.IfType) {
    case IF_TYPE_OTHER:
        wprintf(L"Other\n");
        break;
    case IF_TYPE_ETHERNET_CSMACD:
        wprintf(L"Ethernet\n");
        break;
    case IF_TYPE_IS088025_TOKENRING:
        wprintf(L"Token ring\n");
        break;
    case IF_TYPE_PPP:
        wprintf(L"PPP\n");
        break;
    case IF_TYPE_SOFTWARE_LOOPBACK:
        wprintf(L"Software loopback\n");
        break;
    case IF_TYPE_ATM:
        wprintf(L"ATM\n");
        break;
    case IF_TYPE_IEEE80211:
        wprintf(L"802.11 wireless\n");
        break;
    case IF_TYPE_TUNNEL:
        wprintf(L"Tunnel encapsulation\n");
        break;
    case IF_TYPE_IEEE1394:
        wprintf(L"IEEE 1394 (Firewire)\n");
        break;
    default:
        wprintf(L"Unknown: %d\n", pipRow->InterfaceLuid.Info.IfType);
        break;
    }

    wprintf(L"Interface Index:\t\t %lu\n", pipRow->InterfaceIndex);

    wprintf(L"Prefix Origin:\t\t\t ");
    switch (pipRow->PrefixOrigin) {
    case IpPrefixOriginOther:
        wprintf(L"IpPrefixOriginOther\n");

```

```

        break;
    case IpPrefixOriginManual:
        wprintf(L"IpPrefixOriginManual\n");
        break;
    case IpPrefixOriginWellKnown:
        wprintf(L"IpPrefixOriginWellKnown\n");
        break;
    case IpPrefixOriginDhcp:
        wprintf(L"IpPrefixOriginDhcp\n");
        break;
    case IpPrefixOriginRouterAdvertisement:
        wprintf(L"IpPrefixOriginRouterAdvertisement\n");
        break;
    case IpPrefixOriginUnchanged:
        wprintf(L"IpPrefixOriginUnchanged\n");
        break;
    default:
        wprintf(L"Unknown: %d\n", pipRow->PrefixOrigin);
        break;
    }

    wprintf(L"Suffix Origin:\t\t\t ");
    switch (pipRow->SuffixOrigin) {
    case IpSuffixOriginOther:
        wprintf(L"IpSuffixOriginOther\n");
        break;
    case IpSuffixOriginManual:
        wprintf(L"IpSuffixOriginManual\n");
        break;
    case IpSuffixOriginWellKnown:
        wprintf(L"IpSuffixOriginWellKnown\n");
        break;
    case IpSuffixOriginDhcp:
        wprintf(L"IpSuffixOriginDhcp\n");
        break;
    case IpSuffixOriginLinkLayerAddress:
        wprintf(L"IpSuffixOriginLinkLayerAddress\n");
        break;
    case IpSuffixOriginRandom:
        wprintf(L"IpSuffixOriginRandom\n");
        break;
    case IpSuffixOriginUnchanged:
        wprintf(L"IpSuffixOriginUnchanged\n");
        break;
    default:
        wprintf(L"Unknown: %d\n", pipRow->SuffixOrigin);
        break;
    }

    wprintf(L"Valid Lifetime:\t\t\t 0x%u (%u)\n",
           pipRow->ValidLifetime, pipRow->ValidLifetime);

    wprintf(L"Preferred Lifetime:\t\t\t 0x%u (%u)\n",
           pipRow->PreferredLifetime, pipRow->PreferredLifetime);

```

```

wprintf(L"OnLink PrefixLength:\t\t %lu\n", pipRow->OnLinkPrefixLength);

wprintf(L"Skip As Source:\t\t ");
if (pipRow->SkipAsSource)
    wprintf(L"Yes\n");
else
    wprintf(L"No\n");

wprintf(L"Dad State:\t\t ");
switch (pipRow->DadState) {
case IpDadStateInvalid:
    wprintf(L"IpDadStateInvalid\n");
    break;
case IpDadStateTentative:
    wprintf(L"IpDadStateTentative\n");
    break;
case IpDadStateDuplicate:
    wprintf(L"IpDadStateDuplicate\n");
    break;
case IpDadStateDeprecated:
    wprintf(L"IpDadStateDeprecated\n");
    break;
case IpDadStatePreferred:
    wprintf(L"IpDadStatePreferred\n");
    break;
default:
    wprintf(L"Unknown: %d\n", pipRow->DadState);
    break;
}

wprintf(L"\n");
}

```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateUnicastIpAddressEntry](#)

[DeleteUnicastIpAddressEntry](#)

[GetUnicastIpAddressTable](#)

[IP 帮助程序函数参考](#)

[InitializeUnicastIpAddressEntry](#)

[MIB_UNICASTIPADDRESS_ROW](#)

[MIB_UNICASTIPADDRESS_TABLE](#)

[NotifyUnicastIpAddressChange](#)

[SetUnicastIpAddressEntry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

getUnicastIpAddressTable 函数 (netioapi.h)

项目2023/08/25

GetUnicastIpAddressTable 函数检索本地计算机上的单播 IP 地址表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
GetUnicastIpAddressTable(
    [in] ADDRESS_FAMILY Family,
    [out] PMIB_UNICASTIPADDRESS_TABLE *Table
);
```

parameters

[in] Family

要检索的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_地址系列和PF_协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 *Ws2def.h* 头文件中定义了此成员的可能值。请注意，*Ws2def.h* 头文件会自动包含在 *Winsock2.h* 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	地址系列未指定。指定此参数时，此函数返回包含 IPv4 和 IPv6 条目的单播 IP 地址表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数时，此函数返回仅包含 IPv4 条目的单播 IP 地址表。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数时，此函数返回仅包含 IPv6 条目的单播 IP 地址表。

[out] Table

指向 [MIB_UNICASTIPADDRESS_TABLE](#) 结构的指针，该结构包含本地计算机上的单播 IP 地址条目表。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Table</i> 参数中传递 NULL 指针，或者 <i>Family</i> 参数未指定为AF_INET、AF_INET6或AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存资源不足，无法完成操作。
ERROR_NOT_FOUND	找不到元素。如果未找到 <i>Family</i> 参数中指定的单播 IP 地址条目，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且已在 <i>Family</i> 参数中指定了AF_INET，则返回此错误。如果本地计算机上没有 IPv6 堆栈，并且已在 <i>Family</i> 参数中指定了AF_INET6，则也会返回此错误。在不支持此函数的 Windows 版本上也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[GetUnicastIpAddressTable](#) 函数是在 Windows Vista 及更高版本上定义的。

The

[GetUnicastIpAddressTable](#) 函数枚举本地系统上的单播 IP 地址，并在 [MIB_UNICASTIPADDRESS_TABLE](#) 结构中返回此信息。

单播 IP 地址条目在 *Table* 参数指向的缓冲区中的[MIB_UNICASTIPADDRESS_TABLE](#)结构中返回。[MIB_UNICASTIPADDRESS_TABLE](#) 结构包含单播 IP 地址条目计数和每个单播 IP 地址条目的[MIB_UNICASTIPADDRESS_ROW](#)结构数组。当不再需要这些返回的结构时，通过调用 [FreeMibTable](#) 释放内存。

Family 参数必须初始化为 AF_INET、AF_INET6 或 AF_UNSPEC。

请注意，*Table* 参数指向的返回[MIB_UNICASTIPADDRESS_TABLE](#)结构可能包含 *NumEntries* 成员与 [MIB_UNICASTIPADDRESS_TABLE 结构的 Table 成员中的第一个 MIB_UNICASTIPADDRESS_ROW](#) 数组条目之间的对齐填充。
[MIB_UNICASTIPADDRESS_ROW](#) 数组条目之间也可能存在对齐的填充。对 [MIB_UNICASTIPADDRESS_ROW](#) 数组条目的任何访问都应假定可能存在填充。

示例

以下示例检索单播 IP 地址表，并打印每个检索到[MIB_UNICASTIPADDRESS_ROW](#) 结构中的一些值。

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <Windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

// Need to link with Iphlpapi.lib and Ws2_32.lib
#pragma comment (lib, "iphlpapi.lib")
#pragma comment (lib, "Ws2_32.lib")

int __cdecl wmain()
{
    // Declare and initialize variables

    unsigned int i;

    DWORD Result = 0;

    WCHAR Ipv4String[16] = { 0 };
    WCHAR Ipv6String[46] = { 0 };

    PMIB_UNICASTIPADDRESS_TABLE pipTable = NULL;

    Result = GetUnicastIpAddressTable(AF_UNSPEC, &pipTable);
    if (Result != NO_ERROR) {
        wprintf(L"GetUnicastIpAddressTable returned error: %ld\n", Result);
```

```

    exit(1);
}

// Print some variables from the rows in the table
wprintf(L"Number of table entries: %d\n\n", pipTable->NumEntries);

for (i = 0; i < pipTable->NumEntries; i++) {
    wprintf(L"AddressFamily[%d]:\t\t", i);

    switch (pipTable->Table[i].Address.si_family) {
        case AF_INET:
            wprintf(L"IPv4\n");
            if (InetNtopW
                (AF_INET, &pipTable->Table[i].Address.Ipv4.sin_addr,
                Ipv4String,
                16) != NULL)
                wprintf(L"IPv4 Address:\t\t\t %ws\n", Ipv4String);
            break;
        case AF_INET6:
            wprintf(L"IPv6\n");
            if (InetNtopW
                (AF_INET6, &pipTable->Table[i].Address.Ipv6.sin6_addr,
                Ipv6String, 46) != NULL)
                wprintf(L"IPv6 Address:\t\t\t %ws\n", Ipv6String);
            break;
        default:
            wprintf(L"Other: %d\n", pipTable->Table[i].Address.si_family);
            break;
    }

    wprintf(L"Interface LUID NetLuidIndex[%d]: %lu\n",
           i, pipTable->Table[i].InterfaceLuid.Info.NetLuidIndex);
    wprintf(L"Interface LUID IfType[%d]:\t ", i);
    switch (pipTable->Table[i].InterfaceLuid.Info.IfType) {
        case IF_TYPE_OTHER:
            wprintf(L"Other\n");
            break;
        case IF_TYPE_ETHERNET_CSMACD:
            wprintf(L"Ethernet\n");
            break;
        case IF_TYPE_ISO88025_TOKENRING:
            wprintf(L"Token ring\n");
            break;
        case IF_TYPE_PPP:
            wprintf(L"PPP\n");
            break;
        case IF_TYPE_SOFTWARE_LOOPBACK:
            wprintf(L"Software loopback\n");
            break;
        case IF_TYPE_ATM:
            wprintf(L"ATM\n");
            break;
        case IF_TYPE_IEEE80211:
            wprintf(L"802.11 wireless\n");
            break;
        case IF_TYPE_TUNNEL:
    }
}

```

```
wprintf(L"Tunnel encapsulation\n");
    break;
case IF_TYPE_IEEE1394:
    wprintf(L"IEEE 1394 (Firewire)\n");
    break;
default:
    wprintf(L"Unknown: %d\n",
           pipTable->Table[i].InterfaceLuid.Info.IfType);
    break;
}

wprintf(L"Interface Index[%d]:\t\t %lu\n",
       i, pipTable->Table[i].InterfaceIndex);

wprintf(L"Prefix Origin[%d]:\t\t ", i);
switch (pipTable->Table[i].PrefixOrigin) {
case IpPrefixOriginOther:
    wprintf(L"IpPrefixOriginOther\n");
    break;
case IpPrefixOriginManual:
    wprintf(L"IpPrefixOriginManual\n");
    break;
case IpPrefixOriginWellKnown:
    wprintf(L"IpPrefixOriginWellKnown\n");
    break;
case IpPrefixOriginDhcp:
    wprintf(L"IpPrefixOriginDhcp\n");
    break;
case IpPrefixOriginRouterAdvertisement:
    wprintf(L"IpPrefixOriginRouterAdvertisement\n");
    break;
case IpPrefixOriginUnchanged:
    wprintf(L"IpPrefixOriginUnchanged\n");
    break;
default:
    wprintf(L"Unknown: %d\n", pipTable->Table[i].PrefixOrigin);
    break;
}

wprintf(L"Suffix Origin[%d]:\t\t ", i);
switch (pipTable->Table[i].SuffixOrigin) {
case IpSuffixOriginOther:
    wprintf(L"IpSuffixOriginOther\n");
    break;
case IpSuffixOriginManual:
    wprintf(L"IpSuffixOriginManual\n");
    break;
case IpSuffixOriginWellKnown:
    wprintf(L"IpSuffixOriginWellKnown\n");
    break;
case IpSuffixOriginDhcp:
    wprintf(L"IpSuffixOriginDhcp\n");
    break;
case IpSuffixOriginLinkLayerAddress:
    wprintf(L"IpSuffixOriginLinkLayerAddress\n");
```

```

        break;
    case IpSuffixOriginRandom:
        wprintf(L"IpSuffixOriginRandom\n");
        break;
    case IpSuffixOriginUnchanged:
        wprintf(L"IpSuffixOriginUnchanged\n");
        break;
    default:
        wprintf(L"Unknown: %d\n", pipTable->Table[i].SuffixOrigin);
        break;
    }

    wprintf(L"Valid Lifetime[%d]:\t\t 0x%x (%u)\n", i,
            pipTable->Table[i].ValidLifetime,
            pipTable->Table[i].ValidLifetime);

    wprintf(L"Preferred Lifetime[%d]:\t\t 0x%x (%u)\n", i,
            pipTable->Table[i].PreferredLifetime,
            pipTable->Table[i].PreferredLifetime);

    wprintf(L"OnLink PrefixLength[%d]:\t\t %lu\n", i,
            pipTable->Table[i].OnLinkPrefixLength);

    wprintf(L"Skip As Source[%d]:\t\t ", i);
    if (pipTable->Table[i].SkipAsSource)
        wprintf(L"Yes\n");
    else
        wprintf(L"No\n");

    wprintf(L"Dad State[%d]:\t\t\t ", i);
    switch (pipTable->Table[i].DadState) {
    case IpDadStateInvalid:
        wprintf(L"IpDadStateInvalid\n");
        break;
    case IpDadStateTentative:
        wprintf(L"IpDadStateTentative\n");
        break;
    case IpDadStateDuplicate:
        wprintf(L"IpDadStateDuplicate\n");
        break;
    case IpDadStateDeprecated:
        wprintf(L"IpDadStateDeprecated\n");
        break;
    case IpDadStatePreferred:
        wprintf(L"IpDadStatePreferred\n");
        break;
    default:
        wprintf(L"Unknown: %d\n", pipTable->Table[i].DadState);
        break;
    }

    wprintf(L"\n");
}

if (pipTable != NULL) {

```

```
        FreeMibTable(pipTable);
        pipTable = NULL;
    }

    exit(0);
}
```

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateUnicastIpAddressEntry](#)

[DeleteUnicastIpAddressEntry](#)

[FreeMibTable](#)

[GetUnicastIpAddressEntry](#)

[IP 帮助程序函数参考](#)

[InitializeUnicastIpAddressEntry](#)

[MIB_UNICASTIPADDRESS_ROW](#)

[MIB_UNICASTIPADDRESS_TABLE](#)

[NotifyStableUnicastIpAddressTable](#)

[NotifyUnicastIpAddressChange](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

getUniDirectionalAdapterInfo 函数 (iphlpapi.h)

项目2023/08/24

`GetUniDirectionalAdapterInfo` 函数检索有关本地计算机上安装的单向适配器的信息。
单向适配器是可以接收数据报，但不能传输数据的适配器。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD GetUniDirectionalAdapterInfo(
    [out] PIP_UNIDIRECTIONAL_ADAPTER_ADDRESS pIPIfInfo,
    [out] PULONG                      dwOutBufLen
);
```

parameters

[out] `pIPIfInfo`

指向 `IP_UNIDIRECTIONAL_ADAPTER_ADDRESS` 结构的指针，该结构接收有关本地计算机上安装的单向适配器的信息。

[out] `dwOutBufLen`

指向 `ULONG` 变量的指针，该变量接收 `pIPIfInfo` 参数指向的结构的大小。

返回值

如果函数成功，则返回值`NO_ERROR`。

如果函数失败，请使用 `FormatMessage` 获取返回错误的消息字符串。

要求

目标平台	Windows
标头	iphlpapi.h

Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IP_UNIDIRECTIONAL_ADAPTER_ADDRESS](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

Icmp6CreateFile 函数 (icmpapi.h)

项目2023/08/24

Icmp6CreateFile 函数打开一个句柄，可在该句柄上发出 IPv6 ICMP 回显请求。

语法

C++

```
IPHLPAPI_DLL_LINKAGE HANDLE Icmp6CreateFile();
```

返回值

Icmp6CreateFile 函数在成功时返回打开的句柄。失败时，函数返回 INVALID_HANDLE_VALUE。调用 [GetLastError](#) 函数以获取扩展错误信息。

注解

Icmp6CreateFile 函数打开一个句柄，可在该句柄上发出 IPv6 ICMP 回显请求。

[Icmp6SendEcho2](#) 函数用于发送 IPv6 ICMP 回显请求。[Icmp6ParseReplies](#) 函数用于分析 IPv6 ICMP 回复。[IcmpCloseHandle](#) 函数用于关闭 Icmp6CreateFile 函数打开的 ICMP 句柄。

对于 IPv4，请使用 [IcmpCreateFile](#) 函数。

对于 IPv4，请使用 [IcmpCreateFile](#)、[IcmpSendEcho](#)、[IcmpSendEcho2](#)、[IcmpSendEcho2Ex](#) 和 [IcmpParseReplies](#) 函数。

请注意，*Iphlpapi.h* 头文件的 include 指令必须放在 *Icmpapi.h* 头文件之前。

示例

以下示例打开一个句柄，可在该句柄上发出 IPv6 ICMP 回显请求。

C++

```
#include <windows.h>
#include <stdio.h>
#include <iphlpapi.h>
#include <icmpapi.h>
#pragma comment(lib, "IPHLPIA.lib")
```

```
void main()
{
    HANDLE hIcmpFile;

    hIcmpFile = Icmp6CreateFile();
    if (hIcmpFile == INVALID_HANDLE_VALUE) {
        printf("\tUnable to open handle.\n");
        printf("Icmp6Createfile returned error: %ld\n", GetLastError());
    }
    else
        printf("\tHandle created.\n");
}
```

要求

最低受支持的客户端	Windows XP [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2003 [桌面应用 UWP 应用]
目标平台	Windows
标头	icmpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetLastError](#)

[Icmp6ParseReplies](#)

[Icmp6SendEcho2](#)

[IcmpCloseHandle](#)

[IcmpCreateFile](#)

[IcmpParseReplies](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

Icmp6ParseReplies 函数 (icmpapi.h)

项目2023/08/24

Icmp6ParseReplies 函数分析提供的回复缓冲区，如果找到，则返回 IPv6 ICMPv6 回显响应回复。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD Icmp6ParseReplies(
    [in] LPVOID ReplyBuffer,
    [in] DWORD   ReplySize
);
```

parameters

[in] ReplyBuffer

指向传递到 [Icmp6SendEcho2](#) 函数的缓冲区的指针。此参数指向用于保存响应 的 [ICMPV6_ECHO_REPLY](#) 结构。

[in] ReplySize

ReplyBuffer 参数指向的缓冲区的大小（以字节为单位）。

返回值

成功时，Icmp6ParseReplies 函数返回 1。在这种情况下，如果目标节点响应或 [IP_TTL_EXPIRED_TRANSIT](#)，则 *ReplyBuffer* 参数指向的 [ICMPV6_ECHO_REPLY](#) 结构中的 Status 成员将 [IP_SUCCESS](#)。

如果返回值为零，则可通过 [GetLastError](#) 获取扩展错误信息。

返回代码	说明
ERROR_GEN_FAILURE	发生了一般故障。如果 <i>ReplyBuffer</i> 参数为 NULL 指针或 <i>ReplySize</i> 参数为零，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

IPv6 使用 [Icmp6ParseReplies](#) 函数分析 ICMPv6 回显请求产生的回复。

[Icmp6ParseReplies](#) 函数分析以前传递给 [Icmp6SendEcho2](#) 函数的回复缓冲区。仅对 [Icmp6SendEcho2](#) 函数使用 [Icmp6ParseReplies](#) 函数。

[Icmp6ParseReplies](#) 函数不能在以前传递给 IPv4 的 [IcmpSendEcho](#) 或 [IcmpSendEcho2](#) 的回复缓冲区上使用。

对于 IPv4，请使用 [IcmpCreateFile](#)、[IcmpSendEcho](#)、[IcmpSendEcho2](#)、[IcmpSendEcho2Ex](#) 和 [IcmpParseReplies](#) 函数。

请注意，*Iphlpapi.h* 头文件的 include 指令必须放在 *Icmpapi.h* 头文件之前。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	icmpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetLastError](#)

[ICMPV6_ECHO_REPLY](#)

[ICMP_ECHO_REPLY](#)

[Icmp6CreateFile](#)

[Icmp6Sendecho2](#)

[IcmpCloseHandle](#)

[IcmpCreateFile](#)

[IcmpParseReplies](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

[IcmpSendEcho2Ex](#)

反馈

此页面是否有帮助？

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

icmp6SendEcho2 函数 (icmpapi.h)

项目2023/10/12

`Icmp6SendEcho2` 函数发送 IPv6 ICMPv6 回显请求，如果 *Event* 或 *ApcRoutine* 为非 NULL) 或指定超时后返回，则立即返回 (。 *ReplyBuffer* 包含 IPv6 ICMPv6 回显响应 (如果有)。

语法

C++

```
IPHLAPI_DLL_LINKAGE DWORD Icmp6SendEcho2(
    [in]          HANDLE           IcmpHandle,
    [in, optional] HANDLE          Event,
    [in, optional] PIO_APC_ROUTINE ApcRoutine,
    [in, optional] PVOID           ApcContext,
    [in]          sockaddr_in6   *SourceAddress,
    [in]          sockaddr_in6   *DestinationAddress,
    [in]          LPVOID           RequestData,
    [in]          WORD             RequestSize,
    [in, optional] PIP_OPTION_INFORMATION RequestOptions,
    [out]         LPVOID           ReplyBuffer,
    [in]          DWORD            ReplySize,
    [in]          DWORD            Timeout
);
```

参数

[in] `IcmpHandle`

由 `Icmp6CreateFile` 返回的打开句柄。

[in, optional] `Event`

每当 ICMPv6 响应到达时要发出信号的事件。如果指定此参数，则需要有效事件对象的句柄。使用 `CreateEvent` 或 `CreateEventEx` 函数创建此事件对象。

有关使用事件的详细信息，请参阅 [事件对象](#)。

[in, optional] `ApcRoutine`

当调用线程位于可警报线程中且 ICMPv6 回复到达时调用的例程。在 Windows Vista 及更高版本上，必须定义 `PIO_APC_ROUTINE_DEFINED` 以强制此参数的数据类型 `PIO_APC_ROUTINE` 而不是 `FARPROC`。

在 Windows Server 2003 和 Windows XP 上，不得定义 `PIO_APC_ROUTINE_DEFINED` 以强制将此参数的数据类型强制为 `FARPROC`。

[in, optional] `ApcContext`

每当 ICMPv6 响应到达或发生错误时，传递给 `ApcRoutine` 参数中指定的回调例程的可选参数。

[in] `SourceAddress`

要对其发出回显请求的 IPv6 源地址，采用 `sockaddr` 结构的形式。

[in] `DestinationAddress`

回显请求的 IPv6 目标地址，采用 `sockaddr` 结构的形式。

[in] `requestData`

指向缓冲区的指针，该缓冲区包含要发送的请求中的数据。

[in] `RequestSize`

`requestData` 参数指向的请求数据缓冲区的大小（以字节为单位）。

[in, optional] `RequestOptions`

指向请求的 IPv6 标头选项的指针，形式为 `IP_OPTION_INFORMATION` 结构。在 64 位平台上，此参数采用 `IP_OPTION_INFORMATION32` 结构的形式。

如果不需要指定 IP 标头选项，此参数可能为 NULL。

注意 在 Windows Server 2003 和 Windows XP 上，`RequestOptions` 参数不是可选的，并且不能为 NULL，并且仅使用 `Ttl` 和 `Flags` 成员。

[out] `ReplyBuffer`

指向用于保存请求答复的缓冲区的指针。返回时，缓冲区包含 `ICMPV6_ECHO_REPLY` 结构，后跟来自 ICMPv6 回送响应回复数据的消息正文。缓冲区必须足够大，以容纳至少一个 `ICMPV6_ECHO_REPLY` 结构加上 `RequestSize` 参数中指定的数据字节数。此缓冲区还应足够大，以 (ICMP 错误消息的大小再容纳 8 个字节的数据，) 外加 `IO_STATUS_BLOCK` 结构的空间。

[in] `ReplySize`

ReplyBuffer 参数指向的回复缓冲区的大小（以字节为单位）。此缓冲区应足够大，以容纳至少一个 [ICMPV6_ECHO_REPLY](#) 结构加上 *RequestSize* 字节的数据。此缓冲区还应足够大，以 (ICMP 错误消息的大小再容纳 8 个字节的数据，) 外加 [IO_STATUS_BLOCK](#) 结构的空间。

[in] *Timeout*

等待答复的时间（以毫秒为单位）。仅当同步调用 [Icmp6SendEcho2](#) 函数时，才使用此参数。因此，如果 *ApcRoutine* 或 *Event* 参数不为 `NULL`，则不使用此参数。

返回值

同步调用时，返回在 *ReplyBuffer* 中接收和存储的答复数。

异步调用时，通过返回 [ERROR_IO_PENDING](#) 指示操作正在进行中。稍后，当 *Event* 参数中指定的事件发出信号时，或者调用 *ApcRoutine* 参数中的回调函数时，可以检索回复数结果。

如果（同步或异步）答复数值为零，则对于扩展错误信息，请调用 [GetLastError](#)。

如果函数失败，则 [GetLastError](#) 返回的扩展错误代码可以是以下值之一。

返回代码	说明
<code>ERROR_CALL_NOT_IMPLEMENTED</code>	此系统不支持该功能。
<code>ERROR_INSUFFICIENT_BUFFER</code>	传递给系统调用的数据区域太小。如果 <i>ReplySize</i> 参数指示 <i>ReplyBuffer</i> 参数指向的缓冲区太小，则返回此错误。
<code>ERROR_INVALID_PARAMETER</code>	其中一个参数无效。如果 <i>IcmpHandle</i> 参数包含无效句柄，则返回此错误。
<code>ERROR_IO_PENDING</code>	操作正在进行中。此值由对 Icmp6SendEcho2 的成功异步调用返回，并不指示出现错误。
<code>ERROR_NOT_ENOUGH_MEMORY</code>	内存不足，无法处理此命令。
<code>ERROR_NOT_SUPPORTED</code>	不支持该请求。如果本地计算机上没有 IPv6 堆栈，则返回此错误。
<code>IP_BUF_TOO_SMALL</code>	在 <i>ReplySize</i> 参数中指定的 <i>ReplyBuffer</i> 的大小太小。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

如果 *ApcRoutine* 或 *Event* 参数为 NULL，则将同步调用 **Icmp6SendEcho2** 函数。以同步方式调用时，返回值包含等待 *Timeout* 参数中指定的时间后在 *ReplyBuffer* 中接收和存储的答复数。如果返回值为零，请调用 [GetLastError](#) 以获取扩展错误信息。

指定 *ApcRoutine* 或 *Event* 参数时，将异步调用 **Icmp6SendEcho2** 函数。异步调用时，需要 *ReplyBuffer* 和 *ReplySize* 参数才能接受响应。将 ICMP 响应数据复制到提供的 *ReplyBuffer* 中，当) 指定 *Event* 参数或) 指定 *ApcRoutine* 参数时，(调用回调函数时，应用程序 (发出信号。应用程序必须使用 **Icmp6ParseReplies** 函数分析 *ReplyBuffer* 参数指向的数据。

如果指定了 *Event* 参数，则异步调用 **Icmp6SendEcho2** 函数。每当 ICMPv6 响应到达时，*Event* 参数中指定的事件将发出信号。使用 [CreateEvent](#) 函数创建此事件对象。

如果指定了 *ApcRoutine* 参数，则以异步方式调用 **Icmp6SendEcho2** 函数。*ApcRoutine* 参数应指向用户定义的回调函数。每当 ICMPv6 响应到达时，将调用 *ApcRoutine* 参数中指定的回调函数。对 *ApcRoutine* 参数中指定的回调函数的调用进行序列化。

如果同时指定了 *Event* 和 *ApcRoutine* 参数，则每当 ICMPv6 响应到达时，*Event* 参数中指定的事件将发出信号，但 *ApcRoutine* 参数中指定的回调函数将被忽略。

在 Windows Vista 及更高版本上，任何使用 *ApcRoutine* 参数异步调用 **Icmp6SendEcho2** 函数的应用程序都必须定义 **PIO_AP_C_ROUTINE_DEFINED**，以强制 *ApcRoutine* 参数的数据类型 **PIO_AP_C_ROUTINE** 而不是 **FARPROC**。

请注意，必须先定义 **PIO_AP_C_ROUTINE_DEFINED，然后才能包含 *Icmpapi.h* 头文件。**

在 Windows Vista 及更高版本中，必须将 *ApcRoutine* 指向的回调函数定义为使用以下语法的 **VOID** 类型的函数：

syntax

```
typedef
VOID WINAPI
(*PIO_AP_C_ROUTINE) (
    IN PVOID ApcContext,
    IN PIO_STATUS_BLOCK IoStatusBlock,
    IN ULONG Reserved
);
```

在 Windows Vista 及更高版本上，传递给回调函数的参数包括：

参数	说明
IN PVOID ApcContext	传递给 <code>Icmp6SendEcho2</code> 函数的 <code>ApcContext</code> 参数。 应用程序可以使用此参数来标识回调函数正在响应的 <code>Icmp6SendEcho2</code> 请求。
IN PIO_STATUS_BLOCK IoStatusBlock	指向 <code>IO_STATUS_BLOCK</code> 的指针。 此变量包含最终完成状态和有关操作的信息。 在回复中实际收到的字节数在 <code>IO_STATUS_BLOCK</code> 结构的 <code>Information</code> 成员中返回。 <code>IO_STATUS_BLOCK</code> 结构在 <code>Wdm.h</code> 头文件中定义。
在 ULONG 保留	此参数为保留参数。

在 Windows Server 2003 和 Windows XP 上，任何使用 `ApcRoutine` 参数异步调用 `Icmp6SendEcho2` 函数的应用程序不得定义 `PIO_APC_ROUTINE_DEFINED` 强制将 `ApcRoutine` 参数的数据类型强制转换为 `FARPROC`，而不是 `PIO_APC_ROUTINE`。

在 Windows Server 2003 和 Windows XP 上，必须使用以下语法将 `ApcRoutine` 指向的回调函数定义为 `VOID` 类型的函数：

syntax

```
typedef
VOID WINAPI
(*FARPROC) (
    IN PVOID ApcContext,
);
```

在 Windows Server 2003 和 Windows XP 上，传递给回调函数的参数包括：

参数	说明
IN PVOID ApcContext	传递给 <code>Icmp6SendEcho2</code> 函数的 <code>ApcContext</code> 参数。 应用程序可以使用此参数来标识回调函数正在响应的 <code>Icmp6SendEcho2</code> 请求。

必须在与调用 `Icmp6SendEcho2` 函数的应用程序相同的进程中实现 `ApcRoutine` 参数中指定的回调函数。 如果回调函数位于单独的 DLL 中，则应在调用 `Icmp6SendEcho2` 函数之前加载 DLL。

对于 IPv4，请使用 [IcmpCreateFile](#)、[IcmpSendEcho](#)、[IcmpSendEcho2](#)、[IcmpSendEcho2Ex](#) 和 [IcmpParseReplies](#) 函数。

请注意，`Iphlpapi.h` 头文件的 `include` 指令必须放在 `Icmpapi.h` 头文件之前。

要求

最低受支持的客户端	Windows XP [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2003 [桌面应用 UWP 应用]
目标平台	Windows
标头	icmpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateEvent](#)

[CreateEventEx](#)

[事件对象](#)

[GetLastError](#)

[ICMPV6_ECHO_REPLY](#)

[ICMP_ECHO_REPLY](#)

[IP_OPTION_INFORMATION](#)

[Icmp6CreateFile](#)

[Icmp6ParseReplies](#)

[IcmpCloseHandle](#)

[IcmpCreateFile](#)

[IcmpParseReplies](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

[Icmpsendecho2Ex](#)

反馈

此页面是否有帮助?

是

否

icmpCloseHandle 函数 (icmpapi.h)

项目2023/08/24

`IcmpCloseHandle` 函数关闭通过调用 `IcmpCreateFile` 或 `Icmp6CreateFile` 函数打开的句柄。

语法

C++

```
IPHLPAPI_DLL_LINKAGE BOOL IcmpCloseHandle(  
    [in] HANDLE IcmpHandle  
);
```

parameters

[in] `IcmpHandle`

要关闭的句柄。此句柄必须已通过调用 `IcmpCreateFile` 或 `Icmp6CreateFile` 返回。

返回值

如果句柄成功关闭，则返回值为 `TRUE`，否则返回 `值为 FALSE`。调用 `GetLastError` 函数以获取扩展的错误信息。

注解

`IcmpCloseHandle` 函数从 Windows 2000 上的 `Icmp.dll` 导出。`IcmpCloseHandle` 函数从 Windows XP 及更高版本的 `Iphlpapi.dll` 导出。不建议使用此函数进行 Windows 版本检查。需要跨 Windows 2000、Windows XP、Windows Server 2003 及更高 Windows 版本的此功能可移植性的应用程序不应静态链接到 `Icmp.lib` 或 `Iphlpapi.lib` 文件。相反，应用程序应通过调用 `LoadLibrary` 和 `GetProcAddress` 来检查 `Iphlpapi.dll` 中是否存在 `IcmpCloseHandle`。否则，应用程序应通过调用 `LoadLibrary` 和 `GetProcAddress` 来检查 `Icmp.dll` 中是否存在 `IcmpCloseHandle`。

请注意，`Iphlpapi.h` 头文件的 `include` 指令必须放在 `Icmpapi.h` 头文件之前。

示例

以下示例打开和关闭可以发出 ICMP 回送请求的句柄。

C++

```
#include <windows.h>
#include <iphlpapi.h>
#include <icmpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")

void main()
{
    HANDLE hIcmpFile;
    BOOL bRetVal;

    hIcmpFile = IcmpCreateFile();
    if (hIcmpFile == INVALID_HANDLE_VALUE)
        printf("IcmpCreateFile failed with error: %ld\n", GetLastError());
    else
    {
        printf("\tHandle created.\n");

        bRetVal = IcmpCloseHandle(hIcmpFile);
        if (bRetVal)
            printf("\tHandle was closed\n");
        else
            printf("IcmpCloseHandle failed with error: %ld\n", GetLastError());
    }
}
```

要求

最低受支持的客户端 Windows 2000 专业版 [桌面应用 |UWP 应用]

最低受支持的服务器 Windows 2000 Server [桌面应用 |UWP 应用]

目标平台 Windows

标头 icmpapi.h

Library Iphlpapi.lib

DLL	Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上的 Iphlpapi.dll;Windows 2000 Server 和 Windows 2000 专业版上的 Icmp.dll
-----	--

另请参阅

[GetLastError](#)

[Icmp6CreateFile](#)

[Icmp6ParseReplies](#)

[Icmp6SendEcho2](#)

[IcmpCreateFile](#)

[IcmpParseReplies](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

[IcmpSendEcho2Ex](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IcmpCreateFile 函数 (icmpapi.h)

项目2023/08/24

IcmpCreateFile 函数打开一个句柄，可在该句柄上发出 IPv4 ICMP 回显请求。

语法

C++

```
IPHLPAPI_DLL_LINKAGE HANDLE IcmpCreateFile();
```

返回值

IcmpCreateFile 函数在成功时返回一个打开的句柄。失败时，函数返回 INVALID_HANDLE_VALUE。调用 [GetLastError](#) 函数以获取扩展错误信息。

注解

IcmpCreateFile 函数是从 Windows 2000 上的 *icmp.dll* 导出的。IcmpCreateFile 函数是从 Windows XP 及更高版本上的 *iphlpapi.dll* 导出的。建议不要使用此函数进行 Windows 版本检查。需要跨 Windows 2000、Windows XP、Windows Server 2003 及更高 Windows 版本的此功能可移植性的应用程序不应静态链接到 *icmp.lib* 或 *iphlpapi.lib* 文件。相反，应用程序应检查 `icmpCreateFile` 在 *iphlpapi.dll* 中是否存在，并调用 [LoadLibrary](#) 和 [GetProcAddress](#)。否则，应用程序应在调用 [LoadLibrary](#) 和 [GetProcAddress](#) 的 *icmp.dll* 中检查 `IcmpCreateFile`。

对于 IPv6，请使用 [Icmp6CreateFile](#)、[Icmp6SendEcho2](#) 和 [Icmp6ParseReplies](#) 函数。

请注意，*iphlpapi.h* 头文件的 include 指令必须放在 *icmpapi.h* 头文件之前。

示例

以下示例打开可对其发出 ICMP 回显请求的句柄。

C++

```
#include <windows.h>
#include <stdio.h>
#include <iphlpapi.h>
#include <icmpapi.h>
```

```

// Need to link with Iphlapi.lib
#pragma comment(lib, "IPHLAPI.lib")

void main()
{
    HANDLE hIcmpFile;

    hIcmpFile = IcmpCreateFile();
    if (hIcmpFile == INVALID_HANDLE_VALUE) {
        printf("\tUnable to open handle.\n");
        printf("IcmpCreatefile returned error: %ld\n", GetLastError() );
    }
    else {
        printf("\tHandle created.\n");
        // Need to close the handle when done using it
        IcmpCloseHandle(hIcmpFile);
    }
}

```

要求

最低受支持的客户端	Windows 2000 专业版 [桌面应用 UWP 应用]
最低受支持的服务器	Windows 2000 Server [桌面应用 UWP 应用]
目标平台	Windows
标头	icmpapi.h
Library	Iphlpapi.lib
DLL	在 Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上 Iphlpapi.dll;Windows 2000 Server 和 Windows 2000 专业版上的 Icmp.dll

另请参阅

[GetLastError](#)

[Icmp6CreateFile](#)

[Icmp6ParseReplies](#)

[Icmp6Sendecho2](#)

[IcmpCloseHandle](#)

[IcmpParseReplies](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

[IcmpSendEcho2Ex](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

icmpParseReplies 函数 (icmpapi.h)

项目2023/08/24

[IcmpParseReplies](#) 函数分析提供的回复缓冲区，并返回找到的 ICMP 回送请求响应数。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD IcmpParseReplies(
    [in] LPVOID ReplyBuffer,
    [in] DWORD   ReplySize
);
```

parameters

[in] ReplyBuffer

传递给 [IcmpSendEcho2](#) 的缓冲区。这会重写为保存 [ICMP_ECHO_REPLY](#) 结构的数组，其类型 **为**[PICMP_ECHO_REPLY](#)。

在 64 位平台上，将此缓冲区重写为保存 [ICMP_ECHO_REPLY32](#) 结构的数组，其类型 **为**[PICMP_ECHO_REPLY32](#)。

[in] ReplySize

ReplyBuffer 参数指向的缓冲区的大小（以字节为单位）。

返回值

[IcmpParseReplies](#) 函数返回成功时发现的 ICMP 响应数。函数在出错时返回零。调用 [GetLastError](#) 以获取其他错误信息。

注解

[IcmpParseReplies](#) 函数不应用于之前传递给 [IcmpSendEcho](#) 的回复缓冲区。

[IcmpSendEcho](#) 函数在返回给用户之前分析该缓冲区。仅将此函数与 [IcmpSendEcho2](#) 一起使用。

`IcmpParseReplies` 函数从 Windows 2000 上的 `Icmp.dll` 导出。 `IcmpParseReplies` 函数从 Windows XP 及更高版本的 `Iphlpapi.dll` 导出。 不建议使用此函数进行 Windows 版本检查。 需要跨 Windows 2000、Windows XP、Windows Server 2003 及更高 Windows 版本的此功能可移植性的应用程序不应静态链接到 `Icmp.lib` 或 `Iphlpapi.lib` 文件。 相反，应用程序应通过调用 `LoadLibrary` 和 `GetProcAddress` 来检查 `Iphlpapi.dll` 中是否存在 `IcmpParseReplies`。 否则，应用程序应通过调用 `LoadLibrary` 和 `GetProcAddress` 来检查 `Icmp.dll` 中是否存在 `IcmpParseReplies`。

请注意，`Iphlpapi.h` 头文件的 `include` 指令必须放在 `Icmpapi.h` 头文件之前。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	<code>icmpapi.h</code>
Library	<code>Iphlpapi.lib</code>
DLL	Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上的 <code>Iphlpapi.dll</code> ; Windows 2000 Server 和 Windows 2000 专业版上的 <code>Icmp.dll</code>

另请参阅

[GetLastError](#)

[ICMP_ECHO_REPLY](#)

[ICMP_ECHO_REPLY32](#)

[Icmp6CreateFile](#)

[Icmp6ParseReplies](#)

[Icmp6SendEcho2](#)

[IcmpCloseHandle](#)

[IcmpCreateFile](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

[IcmpSendEcho2Ex](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IcmpSendEcho 函数 (icmpapi.h)

项目2023/08/24

IcmpSendEcho 函数发送 IPv4 ICMP 回显请求并返回任何回显响应回复。当超时已过期或回复缓冲区已满时，调用将返回。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD IcmpSendEcho(
    [in]          HANDLE           IcmpHandle,
    [in]          IPAddr          DestinationAddress,
    [in]          LPVOID          RequestData,
    [in]          WORD            RequestSize,
    [in, optional] PIP_OPTION_INFORMATION RequestOptions,
    [out]         LPVOID          ReplyBuffer,
    [in]          DWORD           ReplySize,
    [in]          DWORD           Timeout
);
```

parameters

[in] **IcmpHandle**

IcmpCreateFile 函数返回的打开句柄。

[in] **DestinationAddress**

回显请求的 IPv4 目标地址，采用 **IPAddr** 结构的形式。

[in] **requestData**

指向缓冲区的指针，该缓冲区包含请求中要发送的数据。

[in] **RequestSize**

requestData 参数指向的请求数据缓冲区的大小（以字节为单位）。

[in, optional] **RequestOptions**

指向请求的 IP 标头选项的指针，采用 **IP_OPTION_INFORMATION** 结构的形式。在 64 位平台上，此参数采用 **IP_OPTION_INFORMATION32** 结构的形式。

如果不需要指定 IP 标头选项，此参数可能为 NULL。

[out] ReplyBuffer

一个缓冲区，用于保存对回显请求的任何答复。 返回时，缓冲区包含 ICMP_ECHO_REPLY 结构的数组，后跟答复的选项和数据。 缓冲区应足够大，以容纳至少一个 ICMP_ECHO_REPLY 结构加上 RequestSize 数据字节。

[in] ReplySize

回复缓冲区的分配大小（以字节为单位）。 缓冲区应足够大，以容纳至少一个 ICMP_ECHO_REPLY 结构加上 RequestSize 数据字节。

此缓冲区还应足够大，以容纳 8 个字节的数据，（ICMP 错误消息）的大小。

[in] Timeout

等待回复的时间（以毫秒为单位）。

返回值

IcmpSendEcho 函数返回存储在 ReplyBuffer 中的 ICMP_ECHO_REPLY 结构的数目。 每个回复的状态都包含在 结构中。 如果返回值为零，请调用 [GetLastError](#) 以获取其他错误信息。

如果函数失败，[GetLastError](#) 返回的扩展错误代码可以是以下值之一。

返回代码	说明
ERROR_INSUFFICIENT_BUFFER	传递给系统调用的数据区域太小。 如果 ReplySize 参数指示 ReplyBuffer 参数指向的缓冲区太小，则返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果 IcmpHandle 参数包含无效句柄，则返回此错误。 如果 ReplySize 参数指定的值小于 ICMP_ECHO_REPLY 结构的大小，则也可能返回此错误。
ERROR_NOT_ENOUGH_MEMORY	没有足够的可用内存来完成该操作。
ERROR_NOT_SUPPORTED	不支持该请求。 如果本地计算机上没有 IPv4 堆栈，则返回此错误。
IP_BUF_TOO_SMALL	ReplySize 参数中指定的 ReplyBuffer 的大小太小。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`IcmpSendEcho` 函数将 ICMP 回显请求发送到指定地址，并返回在 `ReplyBuffer` 中接收和存储的答复数。`IcmpSendEcho` 函数是一个同步函数，在等待 `Timeout` 参数中指定的响应时间后返回。如果返回值为零，请调用 `GetLastError` 以获取扩展错误信息。

`IcmpSendEcho2` 和 `IcmpSendEcho2Ex` 函数是支持异步操作的 `IcmpSendEcho` 的增强版本。`IcmpSendEcho2Ex` 函数还允许指定源 IP 地址。此功能在具有多个网络接口的计算机上很有用。

对于 IPv6，请使用 `Icmp6CreateFile`、`Icmp6SendEcho2` 和 `Icmp6ParseReplies` 函数。

`IcmpSendEcho` 函数是从 Windows 2000 上的 `Icmp.dll` 导出的。`IcmpSendEcho` 函数是从 Windows XP 和更高版本的 `Iphlpapi.dll` 导出的。建议不要使用此函数进行 Windows 版本检查。需要跨 Windows 2000、Windows XP、Windows Server 2003 及更高 Windows 版本的此功能可移植性的应用程序不应静态链接到 `Icmp.lib` 或 `Iphlpapi.lib` 文件。相反，应用程序应检查 `icmpSendEcho` 在调用 `LoadLibrary` 和 `GetProcAddress` 的 `Iphlpapi.dll` 中是否存在 `IcmpSendEcho`。否则，应用程序应在调用 `LoadLibrary` 和 `GetProcAddress` 的 `Icmp.dll` 中检查 `IcmpSendEcho`。

请注意，`Iphlpapi.h` 头文件的 `include` 指令必须放在 `Icmpapi.h` 头文件之前。

示例

以下示例将 ICMP 回显请求发送到命令行中指定的 IP 地址，并打印从第一个响应接收的信息。

```
C++  
  
#include <winsock2.h>  
#include <iphlpapi.h>  
#include <icmpapi.h>  
#include <stdio.h>  
  
#pragma comment(lib, "iphlpapi.lib")  
#pragma comment(lib, "ws2_32.lib")  
  
int __cdecl main(int argc, char **argv) {  
  
    // Declare and initialize variables  
  
    HANDLE hIcmpFile;  
    unsigned long ipaddr = INADDR_NONE;  
    DWORD dwRetVal = 0;  
    char SendData[32] = "Data Buffer";  
    LPVOID ReplyBuffer = NULL;  
    DWORD ReplySize = 0;
```

```

// Validate the parameters
if (argc != 2) {
    printf("usage: %s IP address\n", argv[0]);
    return 1;
}

ipaddr = inet_addr(argv[1]);
if (ipaddr == INADDR_NONE) {
    printf("usage: %s IP address\n", argv[0]);
    return 1;
}

hIcmpFile = IcmpCreateFile();
if (hIcmpFile == INVALID_HANDLE_VALUE) {
    printf("\tUnable to open handle.\n");
    printf("IcmpCreatefile returned error: %ld\n", GetLastError() );
    return 1;
}

ReplySize = sizeof(ICMP_ECHO_REPLY) + sizeof(SendData);
ReplyBuffer = (VOID*) malloc(ReplySize);
if (ReplyBuffer == NULL) {
    printf("\tUnable to allocate memory\n");
    return 1;
}

dwRetVal = IcmpSendEcho(hIcmpFile, ipaddr, SendData, sizeof(SendData),
    NULL, ReplyBuffer, ReplySize, 1000);
if (dwRetVal != 0) {
    PICMP_ECHO_REPLY pEchoReply = (PICMP_ECHO_REPLY)ReplyBuffer;
    struct in_addr ReplyAddr;
    ReplyAddr.S_un.S_addr = pEchoReply->Address;
    printf("\tSent icmp message to %s\n", argv[1]);
    if (dwRetVal > 1) {
        printf("\tReceived %ld icmp message responses\n", dwRetVal);
        printf("\tInformation from the first response:\n");
    }
    else {
        printf("\tReceived %ld icmp message response\n", dwRetVal);
        printf("\tInformation from this response:\n");
    }
    printf("\t Received from %s\n", inet_ntoa( ReplyAddr ) );
    printf("\t Status = %ld\n",
        pEchoReply->Status);
    printf("\t Roundtrip time = %ld milliseconds\n",
        pEchoReply->RoundTripTime);
}
else {
    printf("\tCall to IcmpSendEcho failed.\n");
    printf("\tIcmpSendEcho returned error: %ld\n", GetLastError() );
    return 1;
}
return 0;

```

}

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	icmpapi.h
Library	Iphlpapi.lib
DLL	在 Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上 Iphlpapi.dll;Windows 2000 Server 和 Windows 2000 专业版上的 Icmp.dll

另请参阅

[GetLastError](#)

[ICMP_ECHO_REPLY](#)

[IPAddr](#)

[IP_OPTION_INFORMATION](#)

[IP_OPTION_INFORMATION32](#)

[Icmp6CreateFile](#)

[Icmp6ParseReplies](#)

[Icmp6Sendecho2](#)

[IcmpCloseHandle](#)

[IcmpCreateFile](#)

[IcmpParseReplies](#)

IcmpSendEcho2

IcmpSendEcho2Ex

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IcmpSendEcho2 函数 (icmpapi.h)

项目2023/03/26

IcmpSendEcho2 函数发送 IPv4 ICMP 回显请求，如果 *Event* 或 *ApcRoutine* 为非 NULL，则立即返回 (，或在指定的超时后返回。*ReplyBuffer* 包含 ICMP 回显响应 (如果有) 。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD IcmpSendEcho2(
    [in]             HANDLE           IcmpHandle,
    [in, optional]   HANDLE           Event,
    [in, optional]   PIO_APC_ROUTINE ApcRoutine,
    [in, optional]   PVOID            ApcContext,
    [in]             IPAddr          DestinationAddress,
    [in]             LPVOID           RequestData,
    [in]             WORD             RequestSize,
    [in, optional]   PIP_OPTION_INFORMATION RequestOptions,
    [out]            LPVOID           ReplyBuffer,
    [in]             DWORD            ReplySize,
    [in]             DWORD            Timeout
);
```

参数

[in] IcmpHandle

[ICMPCreateFile](#) 函数返回的打开句柄。

[in, optional] Event

当 ICMP 响应到达时，) 最多 (一次要发出信号的事件。如果指定此参数，则它需要有效事件对象的句柄。使用 [CreateEvent](#) 或 [CreateEventEx](#) 函数创建此事件对象。

有关使用事件的详细信息，请参阅 [事件对象](#)。

[in, optional] ApcRoutine

调用线程位于可警报线程中且 ICMPv4 回复到达时调用的例程。必须定义 **PIO_APC_ROUTINE_DEFINED** 才能强制此参数的数据类型 **PIO_APC_ROUTINE**，而不是 **FARPROC**。

[in, optional] ApcContext

传递到 *ApcRoutine* 参数中指定的回调例程的可选参数最多) ICMP 响应或发生错误时 (一次。

[in] DestinationAddress

回显请求的 IPv4 目标，采用 [IPAddr](#) 结构的形式。

[in] requestData

指向缓冲区的指针，该缓冲区包含请求中要发送的数据。

[in] RequestSize

requestData 参数指向的请求数据缓冲区的大小 (以字节为单位)。

[in, optional] RequestOptions

指向请求的 IP 标头选项的指针，采用 [IP_OPTION_INFORMATION](#) 结构的形式。

如果不需要指定 IP 标头选项，此参数可能为 NULL。

[out] ReplyBuffer

指向缓冲区的指针，用于保存对请求的任何答复。返回后，缓冲区包含 [ICMP_ECHO_REPLY](#) 结构数组，后跟选项和数据。

缓冲区必须足够大，以容纳至少一个 [ICMP_ECHO_REPLY](#) 结构，加上 *RequestSize* 数据字节数，以及额外的 8 个字节数据，(ICMP 错误消息) 的大小。

[in] ReplySize

回复缓冲区的分配大小 (以字节为单位)。

缓冲区必须足够大，以容纳至少一个 [ICMP_ECHO_REPLY](#) 结构，加上 *RequestSize* 数据字节数，以及额外的 8 个字节数据，(ICMP 错误消息) 的大小。

[in] Timeout

等待回复的时间 (以毫秒为单位)。

返回值

以同步方式调用时，[IcmpSendEcho2](#) 函数返回在 *ReplyBuffer* 中接收和存储的答复数。如果返回值为零，则对于扩展错误信息，请调用 [GetLastError](#)。

异步调用时, `IcmpSendEcho2` 函数返回零。对 `GetLastError` 的后续调用 `ERROR_IO_PENDING` 返回 **扩展错误代码**, 指示操作正在进行中。当 `Event` 参数中指定的事件或调用 `ApcRoutine` 参数中的回调函数时, 可以稍后检索结果。

如果返回值为零, 则对于扩展错误信息, 请调用 `GetLastError`。

如果函数失败, 则 `GetLastError` 返回的扩展错误代码可以是以下值之一。

返回代码	说明
<code>ERROR_INVALID_PARAMETER</code>	传递给函数的参数无效。如果 <code>IcmpHandle</code> 参数包含无效句柄, 则返回此错误。如果 <code>ReplySize</code> 参数指定的值小于 <code>ICMP_ECHO_REPLY</code> 结构的大小, 则也可能返回此错误。
<code>ERROR_IO_PENDING</code>	操作正在进行中。此值由对 <code>IcmpSendEcho2</code> 的成功异步调用返回, 并不表示出现错误。
<code>ERROR_NOT_ENOUGH_MEMORY</code>	没有足够的可用内存来完成该操作。
<code>ERROR_NOT_SUPPORTED</code>	不支持该请求。如果本地计算机上没有 IPv4 堆栈, 则返回此错误。
<code>IP_BUF_TOO_SMALL</code>	<code>ReplySize</code> 参数中指定的 <code>ReplyBuffer</code> 的大小太小。
其他	使用 <code>FormatMessage</code> 获取返回错误的消息字符串。

备注

如果 `ApcRoutine` 或 `Event` 参数为 `NULL`, 则同步调用 `IcmpSendEcho2` 函数。以同步方式调用时, 返回值包含等待 `Timeout` 参数中指定的时间后在 `ReplyBuffer` 中接收和存储的答复数。如果返回值为零, 则对于扩展错误信息, 请调用 `GetLastError`。

指定 `ApcRoutine` 或 `Event` 参数时, 将异步调用 `IcmpSendEcho2` 函数。异步调用时, 需要 `ReplyBuffer` 和 `ReplySize` 参数才能接受响应。ICMP 响应数据将复制到提供的 `ReplyBuffer`, 当) 指定 `Event` 参数或) 指定 `ApcRoutine` 参数时 (调用回调函数时, 应用程序 (发出信号。应用程序必须使用 `IcmpParseReplies` 函数分析 `ReplyBuffer` 参数指向的数据。

如果指定了 `Event` 参数, 则异步调用 `IcmpSendEcho2` 函数。当 ICMP 响应到达时, 事件参数中指定的事件最多 () 一次。使用 `CreateEvent` 或 `CreateEventEx` 函数创建此事件对象。

如果指定了 `ApcRoutine` 参数, 则异步调用 `IcmpSendEcho2` 函数。`ApcRoutine` 参数应指向用户定义的回调函数。在 ICMP 响应到达时, 最多) 一次调用 `ApcRoutine` 参数中指定的回调函数 (。对 `ApcRoutine` 参数中指定的回调函数的调用进行序列化。

如果同时指定了 *Event* 和 *ApcRoutine* 参数，则在 ICMP 响应到达时，**最多**一次事件(事件，但忽略 *ApcRoutine* 参数中指定的回调函数。

任何使用 *ApcRoutine* 参数异步调用 *IcmpSendEcho2* 函数的应用程序都必须定义 **PIO_AP_C_ROUTINE_DEFINED**，以强制 *ApcRoutine* 参数的数据类型 **PIO_AP_C_ROUTINE** 而不是 **FARPROC**。

① 备注

必须先定义 **PIO_AP_C_ROUTINE_DEFINED，然后才能包含 *Icmpapi.h* 头文件。**

ApcRoutine 指向的回调函数必须使用以下语法定义为 **VOID** 类型的函数：

C++

```
typedef
VOID WINAPI
(*PIO_AP_C_ROUTINE) (
    IN PVOID ApcContext,
    IN PIO_STATUS_BLOCK IoStatusBlock,
    IN ULONG Reserved
);
```

传递给回调函数的参数包括：

参数	说明
IN PVOID ApcContext	传递给 <i>IcmpSendEcho2</i> 函数的 <i>AppContext</i> 参数。应用程序可以使用此参数来标识回调函数正在响应的 <i>IcmpSendEcho2</i> 请求。
IN PIO_STATUS_BLOCK IoStatusBlock	指向 IO_STATUS_BLOCK 的指针。此变量包含最终完成状态和有关操作的信息。在答复中实际收到的字节数在 IO_STATUS_BLOCK 结构的 <i>Information</i> 成员中返回。 IO_STATUS_BLOCK 结构在 <i>Wdm.h</i> 头文件中定义。
在 ULONG 保留	此参数为保留参数。

必须在与调用 *IcmpSendEcho2* 函数的应用程序相同的进程中实现 *ApcRoutine* 参数中指定的回调函数。如果回调函数位于单独的 DLL 中，则应在调用 *IcmpSendEcho2* 函数之前加载 DLL。

IcmpSendEcho2 函数是从 导出的 *Iphlpapi.dll*。

对于 IPv6，请使用 *Icmp6CreateFile*、*Icmp6SendEcho2* 和 *Icmp6ParseReplies* 函数。

头文件的 include 指令必须放在头文件的 include 指令 `Iphlpapi.h` 之前 `Icmpapi.h`。

示例

以下示例同步调用 `IcmpSendEcho2` 函数。该示例将 ICMP 回显请求发送到命令行中指定的 IP 地址，并输出从第一个响应接收的信息。

C++

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <icmpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

int __cdecl main(int argc, char **argv)
{
    // Declare and initialize variables.
    HANDLE hIcmpFile;
    unsigned long ipaddr = INADDR_NONE;
    DWORD dwRetVal = 0;
    DWORD dwError = 0;
    char SendData[] = "Data Buffer";
    LPVOID ReplyBuffer = NULL;
    DWORD ReplySize = 0;

    // Validate the parameters.
    if (argc != 2) {
        printf("usage: %s IP address\n", argv[0]);
        return 1;
    }

    ipaddr = inet_addr(argv[1]);
    if (ipaddr == INADDR_NONE) {
        printf("usage: %s IP address\n", argv[0]);
        return 1;
    }

    hIcmpFile = IcmpCreateFile();
    if (hIcmpFile == INVALID_HANDLE_VALUE) {
        printf("\tUnable to open handle.\n");
        printf("IcmpCreatefile returned error: %ld\n", GetLastError());
        return 1;
    }

    // Allocate space for a single reply.
    ReplySize = sizeof (ICMP_ECHO_REPLY) + sizeof (SendData) + 8;
    ReplyBuffer = (VOID *) malloc(ReplySize);
    if (ReplyBuffer == NULL) {
        printf("\tUnable to allocate memory for reply buffer\n");
    }
}
```

```

    return 1;
}

dwRetVal = IcmpSendEcho2(hIcmpFile, NULL, NULL, NULL,
                        ipaddr, SendData, sizeof(SendData), NULL,
                        ReplyBuffer, ReplySize, 1000);
if (dwRetVal != 0) {
    PICMP_ECHO_REPLY pEchoReply = (PICMP_ECHO_REPLY) ReplyBuffer;
    struct in_addr ReplyAddr;
    ReplyAddr.S_un.S_addr = pEchoReply->Address;
    printf("\tSent icmp message to %s\n", argv[1]);
    if (dwRetVal > 1) {
        printf("\tReceived %ld icmp message responses\n", dwRetVal);
        printf("\tInformation from the first response:\n");
    } else {
        printf("\tReceived %ld icmp message response\n", dwRetVal);
        printf("\tInformation from this response:\n");
    }
    printf("\t Received from %s\n", inet_ntoa(ReplyAddr));
    printf("\t Status = %ld ", pEchoReply->Status);
    switch (pEchoReply->Status) {
    case IP_DEST_HOST_UNREACHABLE:
        printf("(Destination host was unreachable)\n");
        break;
    case IP_DEST_NET_UNREACHABLE:
        printf("(Destination Network was unreachable)\n");
        break;
    case IP_REQ_TIMED_OUT:
        printf("(Request timed out)\n");
        break;
    default:
        printf("\n");
        break;
    }

    printf("\t Roundtrip time = %ld milliseconds\n",
           pEchoReply->RoundTripTime);
} else {
    printf("Call to IcmpSendEcho2 failed.\n");
    dwError = GetLastError();
    switch (dwError) {
    case IP_BUF_TOO_SMALL:
        printf("\tReplyBufferSize too small\n");
        break;
    case IP_REQ_TIMED_OUT:
        printf("\tRequest timed out\n");
        break;
    default:
        printf("\tExtended error returned: %ld\n", dwError);
        break;
    }
    return 1;
}
return 0;
}

```

要求

最低受支持的客户端	Windows 2000 专业版 [桌面应用 UWP 应用]
最低受支持的服务器	Windows 2000 Server [桌面应用 UWP 应用]
目标平台	Windows
标头	icmpapi.h
Library	Iphlpapi.lib
DLL	Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上的 Iphlpapi.dll;Windows 2000 Server 和 Windows 2000 专业版上的Icmp.dll

另请参阅

- [CreateEvent](#)
- [CreateEventEx](#)
- [事件对象](#)
- [GetLastError](#)
- [ICMP_ECHO_REPLY](#)
- [IP_OPTION_INFORMATION](#)
- [IP_OPTION_INFORMATION32](#)
- [IPAddr](#)
- [Icmp6CreateFile](#)
- [Icmp6ParseReplies](#)
- [Icmp6Sendecho2](#)
- [IcmpCloseHandle](#)
- [IcmpCreateFile](#)
- [IcmpParseReplies](#)
- [IcmpSendecho](#)
- [IcmpSendecho2Ex](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

icmpSendEcho2Ex 函数 (icmpapi.h)

项目2023/08/24

`IcmpSendEcho2Ex` 函数发送 IPv4 ICMP 回显请求，如果 *Event* 或 *ApcRoutine* 非 **NULL**，则立即返回 (或在指定的超时后返回。*ReplyBuffer* 包含 ICMP 响应 (如果有)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD IcmpSendEcho2Ex(
    [in]             HANDLE           IcmpHandle,
    [in, optional]   HANDLE           Event,
    [in, optional]   PIO_APC_ROUTINE ApcRoutine,
    [in, optional]   PVOID            ApcContext,
    [in]             IPAddr          SourceAddress,
    [in]             IPAddr          DestinationAddress,
    [in]             LPVOID           RequestData,
    [in]             WORD             RequestSize,
    [in, optional]   PIP_OPTION_INFORMATION RequestOptions,
    [out]            LPVOID           ReplyBuffer,
    [in]             DWORD            ReplySize,
    [in]             DWORD            Timeout
);
```

parameters

[in] `IcmpHandle`

`ICMPCreateFile` 函数返回的打开句柄。

[in, optional] `Event`

每当 ICMP 响应到达时要发出信号的事件。如果指定此参数，则需要有效事件对象的句柄。使用 `CreateEvent` 或 `CreateEventEx` 函数创建此事件对象。

有关使用事件的详细信息，请参阅 [事件对象](#)。

[in, optional] `ApcRoutine`

当调用线程位于可警报线程中并且 ICMP 回复到达时调用的例程。必须定义 `PIO_APC_ROUTINE_DEFINED`，以强制此参数的数据类型 `PIO_APC_ROUTINE` 而不是 `FARPROC`。

[in, optional] ApcContext

每当 ICMP 响应到达或发生错误时，传递给 *ApcRoutine* 参数中指定的回调例程的可选参数。

[in] SourceAddress

要对其发出回显请求的 IPv4 源地址。此地址采用 [IPAddr](#) 结构的形式。

[in] DestinationAddress

回显请求的 IPv4 目标地址。此地址采用 [IPAddr](#) 结构的形式。

[in] requestData

指向缓冲区的指针，该缓冲区包含要发送的请求中的数据。

[in] RequestSize

requestData 参数指向的请求数据缓冲区的大小（以字节为单位）。

[in, optional] RequestOptions

指向请求的 IP 标头选项的指针，采用 [IP_OPTION_INFORMATION](#) 结构的形式。在 64 位平台上，此参数采用 [IP_OPTION_INFORMATION32](#) 结构的形式。

如果不需要指定 IP 标头选项，此参数可能为 **NULL**。

[out] ReplyBuffer

指向缓冲区的指针，用于保存对请求的任何答复。返回时，缓冲区包含 [ICMP_ECHO_REPLY](#) 结构数组，后跟选项和数据。缓冲区必须足够大，以容纳至少一个 [ICMP_ECHO_REPLY](#) 结构加上 *RequestSize* 字节的数据。

此缓冲区还应足够大，以 (ICMP 错误消息的大小再容纳 8 个字节的数据，) 外加 [IO_STATUS_BLOCK](#) 结构的空间。

[in] ReplySize

回复缓冲区的分配大小（以字节为单位）。缓冲区应足够大，以容纳至少一个 [ICMP_ECHO_REPLY](#) 结构和 *RequestSize* 字节的数据。

此缓冲区还应足够大，以 (ICMP 错误消息的大小再容纳 8 个字节的数据，) 外加 [IO_STATUS_BLOCK](#) 结构的空间。

[in] Timeout

等待答复的时间（以毫秒为单位）。

返回值

以同步方式调用时，[IcmpSendEcho2Ex](#) 函数返回在 *ReplyBuffer* 中接收和存储的答复数。如果返回值为零，请调用[GetLastError](#) 以获取扩展错误信息。

以异步方式调用时，[IcmpSendEcho2Ex](#) 函数返回ERROR_IO_PENDING以指示操作正在进行。稍后，当调用 *Event* 参数信号中指定的事件或 *ApcRoutine* 参数中的回调函数时，可以检索结果。

如果返回值为零，请调用[GetLastError](#) 以获取扩展错误信息。

如果函数失败，[GetLastError](#) 返回的扩展错误代码可以是以下值之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>IcmpHandle</i> 参数包含无效句柄，则返回此错误。如果 <i>ReplySize</i> 参数指定的值小于 ICMP_ECHO_REPLY 结构的大小，也可能会返回此错误。
ERROR_IO_PENDING	操作正在进行中。此值由对 IcmpSendEcho2Ex 的成功异步调用返回，并不指示出现错误。
ERROR_NOT_ENOUGH_MEMORY	没有足够的可用内存来完成该操作。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，则返回此错误。
IP_BUF_TOO_SMALL	在 <i>ReplySize</i> 参数中指定的 <i>ReplyBuffer</i> 的大小太小。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[IcmpSendEcho2Ex](#) 函数在 Windows Server 2008 及更高版本上可用。

[IcmpSendEcho2Ex](#) 函数是[IcmpSendEcho2](#) 函数的增强版本，允许用户指定要对其发出 ICMP 请求的 IPv4 源地址。在计算机具有多个网络接口的情况下，[IcmpSendEcho2Ex](#) 函数非常有用。

如果 *ApcRoutine* 或 *Event* 参数为 NULL，则同步调用[IcmpSendEcho2Ex](#) 函数。以同步方式调用时，返回值包含等待 *Timeout* 参数中指定的时间后在 *ReplyBuffer* 中接收和存储的答复数。如果返回值为零，请调用[GetLastError](#) 以获取扩展错误信息。

指定 *ApcRoutine* 或 *Event* 参数时，将异步调用 `IcmpSendEcho2Ex` 函数。异步调用时，需要 *ReplyBuffer* 和 *ReplySize* 参数才能接受响应。将 ICMP 响应数据复制到提供的 *ReplyBuffer* 中，当) 指定 *Event* 参数或) 指定 *ApcRoutine* 参数时，(调用回调函数时，应用程序 (发出信号。应用程序必须使用 `IcmpParseReplies` 函数分析 *ReplyBuffer* 参数指向的数据。

如果指定了 *Event* 参数，则异步调用 `IcmpSendEcho2Ex` 函数。每当 ICMP 响应到达时，*Event* 参数中指定的事件都发出信号。使用 `CreateEvent` 函数创建此事件对象。

如果指定了 *ApcRoutine* 参数，则异步调用 `IcmpSendEcho2Ex` 函数。*ApcRoutine* 参数应指向用户定义的回调函数。每当 ICMP 响应到达时，将调用 *ApcRoutine* 参数中指定的回调函数。对 *ApcRoutine* 参数中指定的回调函数的调用进行序列化。

如果同时指定了 *Event* 和 *ApcRoutine* 参数，则每当 ICMP 响应到达时，*Event* 参数中指定的事件将发出信号，但 *ApcRoutine* 参数中指定的回调函数将被忽略。

任何使用 *ApcRoutine* 参数异步调用 `IcmpSendEcho2Ex` 函数的应用程序都必须定义 `PIO_AP_C_ROUTINE_DEFINED`，以强制 *ApcRoutine* 参数的数据类型 `PIO_AP_C_ROUTINE` 而不是 `FARPROC`。

请注意，必须先定义 `PIO_AP_C_ROUTINE_DEFINED`，然后才能包含 `Icmpapi.h` 头文件。

必须使用以下语法将 *ApcRoutine* 指向的回调函数定义为 `VOID` 类型的函数：

syntax

```
typedef
VOID WINAPI
(*PIO_AP_C_ROUTINE) (
    IN PVOID ApcContext,
    IN PIO_STATUS_BLOCK IoStatusBlock,
    IN ULONG Reserved
);
```

传递给回调函数的参数包括：

参数	说明
IN PVOID <i>ApcContext</i>	传递给 <code>IcmpSendEcho2Ex</code> 函数的 <i>ApcContext</i> 参数。应用程序可以使用此参数来标识回调函数正在响应的 <code>IcmpSendEcho2Ex</code> 请求。

IN PIO_STATUS_BLOCK IoStatusBlock	指向 IO_STATUS_BLOCK 的指针。此变量包含最终完成状态和有关操作的信息。在回复中实际收到的字节数在 IO_STATUS_BLOCK 结构的 Information 成员中返回。 IO_STATUS_BLOCK 结构在 <i>Wdm.h</i> 头文件中定义。
在 ULONG 保留	此参数为保留参数。

ApcRoutine 参数中指定的回调函数必须在调用 **IcmpSendEcho2Ex** 函数的应用程序所在的同一进程中实现。如果回调函数位于单独的 DLL 中，则应在调用 **IcmpSendEcho2Ex** 函数之前加载 DLL。

对于 IPv6，请使用 **Icmp6CreateFile**、**Icmp6SendEcho2** 和 **Icmp6ParseReplies** 函数。

请注意，*Iphlpapi.h* 头文件的 include 指令必须放在 *Icmpapi.h* 头文件之前。

要求

最低受支持的客户端	Windows Vista SP1 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	icmpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateEvent](#)

[CreateEventEx](#)

[事件对象](#)

[GetLastError](#)

[ICMPCreateFile](#)

[ICMP_ECHO_REPLY](#)

IPAddr

IP_OPTION_INFORMATION

IP_OPTION_INFORMATION32

Icmp6CreateFile

Icmp6ParseReplies

Icmp6Sendecho2

IcmpParseReplies

IcmpSendecho

IcmpSendecho2

反馈

此页面是否有帮助?

是

否

在 Microsoft Q&A 获得帮助

if_indextoname 函数 (netioapi.h)

项目2023/08/25

if_indextoname 函数将网络接口的本地索引转换为 ANSI 接口名称。

语法

C++

```
IPHLPAPI_DLL_LINKAGE PCHAR NETIOAPI_API_ if_indextoname(
    [in]  NET_IFINDEX InterfaceIndex,
    [out] PCHAR        InterfaceName
);
```

parameters

[in] InterfaceIndex

网络接口的本地索引。

[out] InterfaceName

指向缓冲区的指针，用于在函数成功返回时保存包含接口名称的 以 NULL 结尾的 ANSI 字符串。此参数指向的缓冲区的长度（以字节为单位）必须等于或大于 IF_NAMESIZE。

返回值

成功后，if_indextoname 返回指向包含接口名称的 以 NULL 结尾的 ANSI 字符串的指针。失败时，返回 NULL 指针。

注解

if_indextoname 函数在 Windows Vista 及更高版本上可用。

if_indextoname 函数将接口索引映射到其相应的名称。此函数设计为 IPv6 的基本套接字扩展的一部分，如 RFC 2553 中的 IETF 中所述。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2553.txt>。

实现 if_indextoname 函数是为了在 Unix 环境中移植应用程序，但 ConvertInterface 函数是首选。if_indextoname 函数可以替换为对 ConvertInterfaceIndexToLuid 函数的调用，

以将接口索引转换为 [NET_LUID](#) 然后调用 [ConvertInterfaceLuidToNameA](#)，将 [NET_LUID](#) 转换为 ANSI 接口名称。

如果 `if_indextoname` 失败并返回 `NULL` 指针，则无法确定错误代码。

InterfaceName 参数指向的缓冲区的长度（以字节为单位）必须等于或大于 `IF_NAMESIZE`，`Netioapi.h` 头文件中声明的值等于 `NDIS_IF_MAX_STRING_SIZE`。在 `Ntddndis.h` 头文件中声明没有终止 `NULL` 的接口名称的最大长度 `NDIS_IF_MAX_STRING_SIZE`。`NDIS_IF_MAX_STRING_SIZE` 定义为 `Ifdef.h` 头文件中定义的 `IF_MAX_STRING_SIZE` 常量。`Ntddndis.h` 和 `Ifdef.h` 头文件将自动包含在 `Iphlpapi.h` 头文件自动包含的 `Netioapi.h` 头文件中。不应直接使用 `Ntddndis.h`、`Ifdef.h` 和 `Netioapi.h` 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	<code>netioapi.h</code> (包括 <code>Iphlpapi.h</code>)
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_nametoindex](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

if_nametoindex 函数 (netioapi.h)

项目2023/08/25

if_nametoindex 函数将网络接口的 ANSI 接口名称转换为接口的本地索引。

语法

C++

```
IPHLPAPI_DLL_LINKAGE NET_IFINDEX NETIOAPI_API_ if_nametoindex(
    [in] PCSTR InterfaceName
);
```

parameters

[in] InterfaceName

指向包含接口名称的以 NULL 结尾的 ANSI 字符串的指针。

返回值

成功后，if_nametoindex 返回本地接口索引。失败时，返回零。

注解

if_nametoindex 函数在 Windows Vista 及更高版本上可用。

if_nametoindex 函数将接口名称映射到其相应的索引中。此函数设计为 IPv6 的基本套接字扩展的一部分，如 RFC 2553 中的 IETF 所述。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2553.txt>。

实现 if_nametoindex 函数是为了使用 Unix 环境实现应用程序的可移植性，但 ConvertInterface 函数是首选函数。if_nametoindex 函数可以替换为对 ConvertInterfaceNameToLuidA 函数的调用，以将 ANSI 接口名称转换为 NET_LUID 然后调用 ConvertInterfaceLuidToIntIndex 以将 NET_LUID 转换为本地接口索引。

如果 if_nametoindex 函数失败并返回零，则无法确定错误代码。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToAlias](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[NET_LUID](#)

[if_indextoname](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

InitializeIpForwardEntry 函数 (netioapi.h)

项目2023/08/25

InitializeIpForwardEntry 函数使用本地计算机上的 IP 路由条目的默认值初始化 MIB_IPFORWARD_ROW2 结构。

语法

C++

```
IPHLPAPI_DLL_LINKAGE VOID NETIOAPI_API_ InitializeIpForwardEntry(
    [out] PMIB_IPFORWARD_ROW2 Row
);
```

parameters

[out] Row

输入时，是指向 IP 路由条目 MIB_IPFORWARD_ROW2 结构条目的指针。返回时，此参数指向 的 MIB_IPFORWARD_ROW2 结构使用 IP 路由条目的默认值进行初始化。

返回值

此函数不返回值。

注解

InitializeIpForwardEntry 函数在 Windows Vista 及更高版本上定义。

InitializeIpForwardEntry 函数必须用于使用 IP 路由条目的默认值初始化 MIB_IPFORWARD_ROW2 结构条目的成员，以便以后与 CreateIpForwardEntry2 函数一起使用。

在输入时，必须向 InitializeIpForwardEntry 传递一个新的 MIB_IPFORWARD_ROW2 结构才能初始化。

输出时，Row 参数指向的 MIB_IPFORWARD_ROW2 结构的 ValidLifetime 和 PreferredLifetime 成员将初始化为 infinite，并且 Loopback、AutoconfigureAddress、

Publish 和 **Immortal** 成员将初始化为 TRUE。此外，**SitePrefixLength**、**Metric** 和 **Protocol** 成员设置为非法值，其他字段初始化为零。

调用 [InitializeIpForwardEntry](#) 后，应用程序可以更改要修改的 MIB_IPFORWARD_ROW2 项中的成员，然后调用 [CreateIpForwardEntry2](#) 以将新的 IP 路由条目添加到本地计算机。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpForwardEntry2](#)

[DeleteIpForwardEntry2](#)

[GetBestRoute2](#)

[GetIpForwardEntry2](#)

[GetIpForwardTable2](#)

[MIB_IPFORWARD_ROW2](#)

[MIB_IPFORWARD_TABLE2](#)

[NotifyRouteChange2](#)

[SetIpForwardEntry2](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

InitializeIpInterfaceEntry 函数 (netioapi.h)

项目2023/08/25

InitializeIpInterfaceEntry 函数使用默认值初始化MIB_IPINTERFACE_ROW项的成员。

语法

C++

```
IPHLPAPI_DLL_LINKAGE VOID NETIOAPI_API_ InitializeIpInterfaceEntry(
    [in, out] PMIB_IPINTERFACE_ROW Row
);
```

parameters

[in, out] Row

指向要初始化 的MIB_IPINTERFACE_ROW 结构的指针。 成功返回后，此参数中的字段使用本地计算机上的接口的默认信息进行初始化。

返回值

此函数不返回值。

注解

InitializeIpInterfaceEntry 函数在 Windows Vista 及更高版本上定义。

输出时，Row 参数指向的 MIB_IPINTERFACE_ROW 结构中的 Family 成员将初始化为 AF_UNSPEC，MIB_IPINTERFACE_ROW 结构中的 InterfaceLuid 成员将初始化为未指定的值，其他字段初始化为零。

InitializeIpInterfaceEntry 函数必须用于使用默认值初始化MIB_IPINTERFACE_ROW结构条目的字段。 然后，应用程序可以更改要修改 的MIB_IPINTERFACE_ROW 条目中的字段，然后调用 SetIpInterfaceEntry 函数。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[GetIpInterfaceEntry](#)

[GetIpInterfaceTable](#)

[IP 帮助程序函数参考](#)

[MIB_IPINTERFACE_ROW](#)

[MIB_IPINTERFACE_TABLE](#)

[SetIpInterfaceEntry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

initializeUnicastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

InitializeUnicastIpAddressEntry 函数使用本地计算机上的单播 IP 地址条目的默认值初始化 MIB_UNICASTIPADDRESS_ROW 结构。

语法

C++

```
IPHLPAPI_DLL_LINKAGE VOID NETIOAPI_API_ InitializeUnicastIpAddressEntry(
    [out] PMIB_UNICASTIPADDRESS_ROW Row
);
```

parameters

[out] Row

输入时，指向单播 IP 地址条目 MIB_UNICASTIPADDRESS_ROW 结构条目的指针。返回时，此参数指向 的 MIB_UNICASTIPADDRESS_ROW 结构使用单播 IP 地址的默认值进行初始化。

返回值

此函数不返回值。

注解

InitializeUnicastIpAddressEntry 函数在 Windows Vista 及更高版本上定义。

InitializeUnicastIpAddressEntry 函数必须用于初始化具有单播 IP 地址默认值 的 MIB_UNICASTIPADDRESS_ROW 结构条目的成员，以便以后与 CreateUnicastIpAddressEntry 函数一起使用。

输入时，必须向 InitializeUnicastIpAddressEntry 传递一个新的 MIB_UNICASTIPADDRESS_ROW 结构才能初始化。

输出时，`Row` 参数指向的 [MIB_UNICASTIPADDRESS_ROW](#) 结构的 `PrefixOrigin` 成员将初始化为 `IpPrefixOriginUnchanged`，`SuffixOrigin` 成员将初始化为 `IpSuffixOriginUnchanged`，`OnLinkPrefixLength` 成员将初始化为非法值。此外，`PreferredLifetime` 和 `ValidLifetime` 成员设置为无限期，`SkipAsSource` 成员设置为 `FALSE`，其他字段初始化为零。

调用 `InitializeUnicastIpAddressEntry` 后，应用程序可以更改要修改的 [MIB_UNICASTIPADDRESS_ROW](#) 条目中的成员，然后调用 `CreateUnicastIpAddressEntry` 以将新的单播 IP 地址添加到本地计算机。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateUnicastIpAddressEntry](#)

[DeleteUnicastIpAddressEntry](#)

[GetUnicastIpAddressEntry](#)

[GetUnicastIpAddressTable](#)

[IP 帮助程序函数参考](#)

[MIB_UNICASTIPADDRESS_ROW](#)

[MIB_UNICASTIPADDRESS_TABLE](#)

[NotifyUnicastIpAddressChange](#)

[SetUnicastIpAddressEntry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

ipReleaseAddress 函数 (iphlpapi.h)

项目2023/08/24

IpReleaseAddress 函数释放以前通过动态主机配置协议 (DHCP) 获取的 IPv4 地址。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD IpReleaseAddress(
    [in] PIP_ADAPTER_INDEX_MAP AdapterInfo
);
```

parameters

[in] AdapterInfo

指向 IP_ADAPTER_INDEX_MAP 结构的指针，该结构指定要发布的 IPv4 地址关联的适配器。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

返回代码	说明
ERROR_INVALID_PARAMETER	其中一个参数无效。如果 AdapterInfo 参数为 NULL，或者 AdapterInfo 参数指向的 IP_ADAPTER_INDEX_MAP 结构的 Name 成员无效，则返回此错误。
ERROR_PROC_NOT_FOUND	在向 DHCP 请求释放 IPv4 地址期间发生异常。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

IpReleaseAddress 函数特定于 IPv4，仅发布以前通过动态主机配置协议 (DHCP) 获取的 IPv4 地址。AdapterInfo 参数指向的 IP_ADAPTER_INDEX_MAP 结构的 Name 成员是用于确定要释放的 DHCP 地址的唯一成员。

`GetInterfaceInfo` 函数在 `IP_INTERFACE_INFO` 结构中返回`IP_ADAPTER_INDEX_MAP`结构数组。 `GetInterfaceInfo` 返回的`IP_INTERFACE_INFO`结构至少包含一个`IP_ADAPTER_INDEX_MAP`结构，即使 `IP_INTERFACE_INFO` 结构的 `NumAdapters` 成员指示未启用具有 IPv4 的网络适配器。当 `GetInterfaceInfo` 返回的 `IP_INTERFACE_INFO` 结构的 `NumAdapters` 成员为零时，`IP_INTERFACE_INFO`结构中返回的单个`IP_ADAPTER_INDEX_MAP`结构的成员的值未定义。

如果 `AdapterInfo` 参数指向的 `IP_ADAPTER_INDEX_MAP` 结构的 `Name` 成员为 `NULL`，则 `IpReleaseAddress` 函数将返回`ERROR_INVALID_PARAMETER`。

没有可用于释放或续订 IPv6 地址的函数。这只能通过执行 `Ipconfig` 命令来完成：

`ipconfig /release6`

`ipconfig /renew6`

示例

以下示例检索本地系统上启用了 IPv4 的网络适配器列表，然后释放并续订列表中的第一个适配器的 IPv4 地址。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

// Before calling IpReleaseAddress and IpRenewAddress we use
// GetInterfaceInfo to retrieve a handle to the adapter

void __cdecl main()
{
    ULONG ulOutBufLen = 0;
    DWORD dwRetVal = 0;
    PIP_INTERFACE_INFO pInfo;

    pInfo = (IP_INTERFACE_INFO *) MALLOC(sizeof(IP_INTERFACE_INFO));

    // Make an initial call to GetInterfaceInfo to get
    // the necessary size into the ulOutBufLen variable
```

```

    if (GetInterfaceInfo(pInfo, &ulOutBufLen) == ERROR_INSUFFICIENT_BUFFER)
    {
        FREE(pInfo);
        pInfo = (IP_INTERFACE_INFO *) MALLOC (ulOutBufLen);
    }

    // Make a second call to GetInterfaceInfo to get the
    // actual data we want
    if ((dwRetVal = GetInterfaceInfo(pInfo, &ulOutBufLen)) == NO_ERROR ) {
        printf("\tAdapter Name: %ws\n", pInfo->Adapter[0].Name);
        printf("\tAdapter Index: %ld\n", pInfo->Adapter[0].Index);
        printf("\tNum Adapters: %ld\n", pInfo->NumAdapters);
    }
    else if (dwRetVal == ERROR_NO_DATA) {
        printf("There are no network adapters with IPv4 enabled on the local
system\n");
        return;
    }
    else {
        LPVOID lpMsgBuf;
        printf("GetInterfaceInfo failed.\n");

        if (FormatMessage(
            FORMAT_MESSAGE_ALLOCATE_BUFFER |
            FORMAT_MESSAGE_FROM_SYSTEM |
            FORMAT_MESSAGE_IGNORE_INSERTS,
            NULL,
            dwRetVal,
            MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
            (LPTSTR) &lpMsgBuf,
            0,
            NULL )) {
            printf("\tError: %s", lpMsgBuf);
        }
        LocalFree( lpMsgBuf );
        return;
    }

    // Call IpReleaseAddress and IpRenewAddress to release and renew
    // the IP address on the first network adapter returned
    // by the call to GetInterfaceInfo.
    if ((dwRetVal = IpReleaseAddress(&pInfo->Adapter[0])) == NO_ERROR) {
        printf("IP release succeeded.\n");
    }
    else {
        printf("IP release failed: %ld\n", dwRetVal);
    }

    if ((dwRetVal = IpRenewAddress(&pInfo->Adapter[0])) == NO_ERROR) {
        printf("IP renew succeeded.\n");
    }
    else {
        printf("IP renew failed: %ld\n", dwRetVal);
    }
}

```

```
// Free memory for IP_INTERFACE_INFO
if (pInfo != NULL) {
    FREE(pInfo);
}
return;
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetInterfaceInfo](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IP_ADAPTER_INDEX_MAP](#)

[IP_INTERFACE_INFO](#)

[IpRenewAddress](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

ipRenewAddress 函数 (iphlpapi.h)

项目2023/08/24

IpRenewAddress 函数续订以前通过动态主机配置协议 (DHCP) 获取的 IPv4 地址的租约。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD IpRenewAddress(
    [in] PIP_ADAPTER_INDEX_MAP AdapterInfo
);
```

parameters

[in] AdapterInfo

指向 IP_ADAPTER_INDEX_MAP 结构的指针，该结构指定要续订的 IP 地址关联的适配器。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，请使用 FormatMessage 获取返回错误的消息字符串。

返回代码	说明
ERROR_INVALID_PARAMETER	其中一个参数无效。如果 AdapterInfo 参数为 NULL，或者 AdapterInfo 参数指向的 PIP_ADAPTER_INDEX_MAP 结构的 Name 成员无效，则返回此错误。
ERROR_PROC_NOT_FOUND	请求 DHCP 续订 IPv4 地址期间发生异常。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

IpRenewAddress 函数特定于 IPv4，仅续订以前通过动态主机配置协议 (DHCP) 获取的 IPv4 地址。AdapterInfo 参数指向的 IP_ADAPTER_INDEX_MAP 结构的 Name 成员是用于确定要续订的 DHCP 地址的唯一成员。

`getInterfaceInfo` 函数在 `IP_INTERFACE_INFO` 结构中返回 `IP_ADAPTER_INDEX_MAP` 结构数组。 `GetInterfaceInfo` 返回的 `IP_INTERFACE_INFO` 结构至少包含一个 `IP_ADAPTER_INDEX_MAP` 结构，即使 `IP_INTERFACE_INFO` 结构的 `NumAdapters` 成员指示未启用具有 IPv4 的网络适配器。当 `GetInterfaceInfo` 返回的 `IP_INTERFACE_INFO` 结构的 `NumAdapters` 成员为零时，`IP_INTERFACE_INFO` 结构中返回的单个 `IP_ADAPTER_INDEX_MAP` 结构的成员的值未定义。

如果 `AdapterInfo` 参数指向的 `IP_ADAPTER_INDEX_MAP` 结构的 `Name` 成员为 `NULL`，则 `IpRenewAddress` 函数将返回 `ERROR_INVALID_PARAMETER`。

没有可用于释放或续订 IPv6 地址的函数。这只能通过执行 `Ipconfig` 命令来完成：

```
ipconfig /release6
```

```
ipconfig /renew6
```

示例

以下示例检索本地系统上启用了 IPv4 的网络适配器列表，然后释放并续订列表中的第一个适配器的 IPv4 地址。

C++

```
#include <windows.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")

/* Note: could also use malloc() and free() */
#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

void main()
{
    // Before calling IpReleaseAddress and IpRenewAddress we use
    // GetInterfaceInfo to retrieve a handle to the adapter

    PIP_INTERFACE_INFO pInfo;
    pInfo = (IP_INTERFACE_INFO *) MALLOC( sizeof(IP_INTERFACE_INFO) );
    ULONG ulOutBufLen = 0;
    DWORD dwRetVal = 0;

    // Make an initial call to GetInterfaceInfo to get
    // the necessary size into the ulOutBufLen variable
    if ( GetInterfaceInfo(pInfo, &ulOutBufLen) == ERROR_INSUFFICIENT_BUFFER )
    {
        FREE(pInfo);
```

```

    pInfo = (IP_INTERFACE_INFO *) MALLOC (ulOutBufLen);
}

// Make a second call to GetInterfaceInfo to get the
// actual data we want
if ((dwRetVal = GetInterfaceInfo(pInfo, &ulOutBufLen)) == NO_ERROR ) {
    printf("\tAdapter Name: %ws\n", pInfo->Adapter[0].Name);
    printf("\tAdapter Index: %ld\n", pInfo->Adapter[0].Index);
    printf("\tNum Adapters: %ld\n", pInfo->NumAdapters);
}
else if (dwRetVal == ERROR_NO_DATA) {
    printf("There are no network adapters with IPv4 enabled on the local
system\n");
    FREE(pInfo);
    pInfo = NULL;
    return;
}
else {
    printf("GetInterfaceInfo failed.\n");
    LPVOID lpMsgBuf;

    if (FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        dwRetVal,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
        (LPTSTR) &lpMsgBuf,
        0,
        NULL )) {
        printf("\tError: %s", lpMsgBuf);
    }
    LocalFree( lpMsgBuf );
    return;
}

// Call IpReleaseAddress and IpRenewAddress to release and renew
// the IP address on the first network adapter returned
// by the call to GetInterfaceInfo.
if ((dwRetVal = IpReleaseAddress(&pInfo->Adapter[0])) == NO_ERROR) {
    printf("IP release succeeded.\n");
}
else {
    printf("IP release failed.\n");
}

if ((dwRetVal = IpRenewAddress(&pInfo->Adapter[0])) == NO_ERROR) {
    printf("IP renew succeeded.\n");
}
else {
    printf("IP renew failed.\n");
}

/* Free allocated memory no longer needed */

```

```
    if (pInfo) {
        FREE(pInfo);
        pInfo = NULL;
    }
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetInterfaceInfo](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IP_ADAPTER_INDEX_MAP](#)

[IP_INTERFACE_INFO](#)

[IpReleaseAddress](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

LookupPersistentTcpPortReservation 函数 (iphlpapi.h)

项目2023/03/14

LookupPersistentTcpPortReservation 函数为本地计算机上的连续 TCP 端口块查找永久性 TCP 端口预留的令牌。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG LookupPersistentTcpPortReservation(
    [in] USHORT StartPort,
    [in] USHORT NumberOfPorts,
    [out] PULONG64 Token
);
```

参数

[in] StartPort

按网络字节顺序排列的起始 TCP 端口号。

[in] NumberOfPorts

保留的 TCP 端口号数。

[out] Token

指向在函数成功时返回的端口预留令牌的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	传递给函数的参数无效。如果在 StartPort 或 NumberOfPorts 参数中传递零，则返回此错误。

ERROR_NOT_FOUND	找不到该元素。如果找不到由 <i>StartPort</i> 和 <i>NumberOfPorts</i> 参数指定的永久性端口块，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

备注

LookupPersistentTcpPortReservation 函数在 Windows Vista 及更高版本上定义。

LookupPersistentTcpPortReservation 函数用于查找 TCP 端口块的永久性预留的令牌。

TCP 端口块的持久预留是通过调用 [CreatePersistentTcpPortReservation](#) 函数创建的。传递给 **LookupPersistentTcpPortReservation** 函数的 **StartPort** 或 **NumberOfPorts** 参数必须与 [CreatePersistentTcpPortReservation](#) 函数创建 TCP 端口块的永久性预留时使用的值匹配。

如果 **LookupPersistentTcpPortReservation** 函数成功，则返回的 *Token* 参数将指向 TCP 端口块的持久端口预留的令牌。请注意，每次重启系统时，TCP 端口块的给定永久性预留的令牌可能会更改。

应用程序可以通过打开 TCP 套接字，然后调用 [WSAIoctl](#) 函数（指定 **SIO_ASSOCIATE_PORT_RESERVATION** IOCTL），并在调用套接字上的 [绑定](#) 函数之前传递预留令牌，从 TCP 端口预留请求端口分配。

示例

以下示例查找永久性 TCP 端口预留，然后创建套接字并从端口预留分配端口。

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <Windows.h>
#include <winsock2.h>
#include <mstcpip.h>
#include <ws2ipdef.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>
```

```
// Need to link with iphlpapi.lib
#pragma comment(lib, "iphlpapi.lib")

// Need to link with Ws2_32.lib for Winsock functions
#pragma comment(lib, "ws2_32.lib")

int wmain(int argc, WCHAR **argv)  {

    // Declare and initialize variables

    int startPort = 0;           // host byte order
    int numPorts = 0;
    USHORT startPortns = 0;      // Network byte order
    ULONG64 resToken = {0};

    unsigned long status = 0;

    WSADATA wsaData = { 0 };
    int iResult = 0;

    SOCKET sock = INVALID_SOCKET;
    int iFamily = AF_INET;
    int iType = SOCK_STREAM;
    int iProtocol = IPPROTO_TCP;

    DWORD bytesReturned = 0;

    // Note that the sockaddr_in struct works only with AF_INET not AF_INET6
    // An application needs to use the sockaddr_in6 for AF_INET6
    sockaddr_in service;
    sockaddr_in sockName;
    int nameLen = sizeof(sockName);

    // Validate the parameters
    if (argc != 3) {
        wprintf(L"usage: %s <Starting Port> <Number of Ports>\n",
               argv[0]);
        wprintf(L"Look up a persistent TCP port reservation\n");
        wprintf(L"Example usage:\n");
        wprintf(L"  %s 5000 20\n", argv[0]);
        wprintf(L"  where StartPort=5000 NumPorts=20");
        return 1;
    }

    startPort = _wtoi(argv[1]);
    if ( startPort < 0 || startPort> 65535) {
        wprintf(L"Starting point must be either 0 or between 1 and
65,535\n");
        return 1;
    }
    startPortns = htons((USHORT) startPort);

    numPorts = _wtoi(argv[2]);
    if (numPorts < 0) {
        wprintf(L"Number of ports must be a positive number\n");
    }
```

```

        return 1;
    }

    status = LookupPersistentTcpPortReservation((USHORT) startPortns,
(USHORT) numPorts, &resToken);
    if( status != NO_ERROR )
    {
        wprintf(L"LookupPersistentTcpPortReservation returned error: %ld\n",
               status);
        return 1;
    }

    wprintf(L"LookupPersistentTcpPortReservation call succeeded\n");
    wprintf(L" Token = %I64d\n", resToken);

    // Comment out this block if you don't want to create a socket and
associate it with the
    // persistent reservation

    // Initialize Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult != 0) {
    wprintf(L"WSAStartup failed with error = %d\n", iResult);
    return 1;
}

sock = socket(iFamily, iType, iProtocol);
if (sock == INVALID_SOCKET)
    wprintf(L"socket function failed with error = %d\n",
WSAGetLastError());
else {
    wprintf(L"socket function succeeded\n");

    iResult =
        WSAIoctl(sock, SIO_ASSOCIATE_PORT_RESERVATION, (LVOID) &
resToken,
                  sizeof (ULONG64), NULL, 0, &bytesReturned, NULL, NULL);
    if (iResult != 0) {
        wprintf
            (L"WSAIoctl(SIO_ASSOCIATE_PORT_RESERVATION) failed with
error = %d\n",
             WSAGetLastError());
    } else {
        wprintf(L"WSAIoctl(SIO_ASSOCIATE_PORT_RESERVATION) succeeded,
bytesReturned = %u\n",
               bytesReturned);

        service.sin_family = AF_INET;
        service.sin_addr.s_addr = INADDR_ANY;
        service.sin_port = 0;

        iResult = bind(sock, (SOCKADDR*) &service, sizeof(service) );
        if (iResult == SOCKET_ERROR)
            wprintf(L"bind failed with error = %d\n",
WSAGetLastError());
    }
}
}

```

```

        else {
            wprintf(L"bind succeeded\n");
            iResult = getsockname(sock, (SOCKADDR*) &sockName,
&nameLen);
            if (iResult == SOCKET_ERROR)
                wprintf(L"getsockname failed with error = %d\n",
WSAGetLastError() );
            else {
                wprintf(L"getsockname succeeded\n");
                wprintf(L"Port number allocated = %u\n",
ntohs(sockName.sin_port) );
            }
        }

        if (sock != INVALID_SOCKET) {
            iResult = closesocket(sock);
            if (iResult == SOCKET_ERROR) {
                wprintf(L"closesocket failed with error = %d\n",
WSAGetLastError());
            }
        }
    }
    WSACleanup();

    return 0;
}

```

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[CreatePersistentTcpPortReservation](#)

[CreatePersistentUdpPortReservation](#)

[DeletePersistentTcpPortReservation](#)

[DeletePersistentUdpPortReservation](#)

[LookupPersistentUdpPortReservation](#)

[SIO_ASSOCIATE_PORT_RESERVATION](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

LookupPersistentUdpPortReservation 函数 (iphlpapi.h)

项目2023/08/24

LookupPersistentUdpPortReservation 函数为本地计算机上的连续 TCP 端口块查找持久 UDP 端口预留的令牌。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG LookupPersistentUdpPortReservation(
    [in] USHORT StartPort,
    [in] USHORT NumberOfPorts,
    [out] PULONG64 Token
);
```

parameters

[in] StartPort

按网络字节顺序表示的起始 UDP 端口号。

[in] NumberOfPorts

保留的 UDP 端口号数。

[out] Token

指向如果函数成功，则返回的端口预留令牌的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 StartPort 或 NumberOfPorts 参数中传递零，则返回此错误。

ERROR_NOT_FOUND	找不到该元素。如果找不到由 <i>StartPort</i> 和 <i>NumberOfPorts</i> 参数指定的永久性端口块，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

LookupPersistentUdpPortReservation 函数在 Windows Vista 及更高版本上定义。

LookupPersistentUdpPortReservation 函数用于查找令牌，以获取 UDP 端口块的持久预留。

UDP 端口块的持久预留是通过调用 [CreatePersistentUdpPortReservation](#) 函数创建的。传递给 **LookupPersistentUdpPortReservation** 函数的 *StartPort* 或 *NumberOfPorts* 参数必须与 [CreatePersistentUdpPortReservation](#) 函数创建 TCP 端口块的永久性预留时使用的值匹配。

如果 **LookupPersistentUdpPortReservation** 函数成功，则返回的 *Token* 参数将指向 UDP 端口块的持久端口预留的令牌。请注意，每次重启系统时，TCP 端口块的给定永久性预留的令牌可能会更改。

应用程序可以通过打开 UDP 套接字，然后调用 [WSAOctl](#) 函数（指定 [SIO_ASSOCIATE_PORT_RESERVATION](#) IOCTL），并在调用套接字上的 [绑定](#) 函数之前传递预留令牌，从 UDP 端口预留请求端口分配。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[CreatePersistentTcpPortReservation](#)

[CreatePersistentUdpPortReservation](#)

[DeletePersistentTcpPortReservation](#)

[DeletePersistentUdpPortReservation](#)

[LookupPersistentTcpPortReservation](#)

[SIO_ASSOCIATE_PORT_RESERVATION](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

NhpAllocateAndGetInterfaceInfoFromStack 函数 (iphlpapi.h)

项目2023/08/24

[此函数在 Windows Vista 中不再可用。 请改用 [GetAdaptersAddresses](#) 函数和关联的 [IP_ADAPTER_ADDRESSES](#) 结构。]

NhpAllocateAndGetInterfaceInfoFromStack 函数获取有关本地计算机的适配器信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD NhpAllocateAndGetInterfaceInfoFromStack(
    IP_INTERFACE_NAME_INFO **ppTable,
    PDWORD             pdwCount,
    BOOL               bOrder,
    HANDLE              hHeap,
    DWORD               dwFlags
);
```

parameters

`ppTable`

包含有关本地系统上每个适配器的信息 [的IP_INTERFACE_NAME_INFO](#) 结构的数组。 数组包含系统上每个适配器的一个元素。

`pdwCount`

ppTable 数组中的元素数。

`bOrder`

如果 为 TRUE，则 *ppTable* 数组中的元素按增加索引值进行排序。

`hHeap`

一个句柄，指定应从中分配 *ppTable* 的堆。 此参数可以是调用 [GetProcessHeap](#) 函数返回的进程堆，也可以是调用 [HeapCreate](#) 函数创建的专用堆。

`dwFlags`

为 *ppTable* 分配内存时要传递给 [HeapAlloc](#) 函数的一组标志。有关详细信息，请参阅 [HeapAlloc](#) 函数。

返回值

成功完成后返回 [ERROR_SUCCESS](#)。

注解

在 Microsoft Windows 软件开发工具包 (SDK) 中，[NhpAllocateAndGetInterfaceInfoFromStack](#) 函数在 Windows 2000 上定义了 Service Pack 1 (SP1) 及更高版本。编译应用程序时，如果目标平台是具有 SP1 的 Windows 2000 和更高版本 (`NTDDI_VERSION >= NTDDI_WIN2KSP1`、`_WIN32_WINNT >= 0x0500` 或 `WINVER >= 0x0500`)，则会定义 [NhpAllocateAndGetInterfaceInfoFromStack](#)。

要求

最低受支持的客户端	具有 SP1 的 Windows XP、Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows Server 2003、Windows 2000 Server SP1 [仅限桌面应用]
目标平台	Windows
标头	<code>iphlpapi.h</code>
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

另请参阅

[GetAdaptersAddresses](#)

[GetProcessHeap](#)

[HeapCreate](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

NotifyAddrChange 函数 (iphlpapi.h)

项目2023/08/24

只要表中发生了将 IPv4 地址映射到接口的更改， NotifyAddrChange 函数就会向调用方发送通知。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD NotifyAddrChange(
    [out] PHANDLE     Handle,
    [in]  LPOVERLAPPED overlapped
);
```

parameters

[out] Handle

指向 HANDLE 变量的指针，该变量接收文件句柄以用于 [对 GetOverlappedResult 函数的后续调用](#)。

警告 不要关闭此句柄，也不要将其与完成端口相关联。

[in] overlapped

指向 OVERLAPPED 结构的指针，它通知调用方表中将 IP 地址映射到接口的任何更改。

返回值

如果函数成功，则如果调用方为句柄参数和重叠参数指定 NULL，则返回值NO_ERROR。如果调用方指定非 NULL 参数，则成功返回值ERROR_IO_PENDING。

如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

返回代码	说明
ERROR_CANCELLED	上下文正在取消注册，因此立即取消了调用。

ERROR_INVALID_PARAMETER	传递的参数无效。如果 句柄 参数和 重叠 参数都不为 NULL，但调用进程无法写入输入参数指定的内存，则返回此错误。如果客户端已发出更改通知请求，则也会返回此错误，因此此重复请求将失败。
ERROR_NOT_ENOUGH_MEMORY	内存不足，无法完成操作。
ERROR_NOT_SUPPORTED	此错误在不支持此函数的 Windows 版本（如 Windows 98/95 和 Windows NT 4.0）上返回。

注解

The

可以通过两种方式调用 **NotifyAddrChange** 函数：

- 同步方法
- 异步方法

如果调用方为 **句柄** 参数和 **重叠** 参数指定 NULL，则对 **NotifyAddrChange** 的调用是同步的，并且将阻止，直到发生 IP 地址更改。在这种情况下，如果发生更改，则 **NotifyAddrChange** 函数完成以指示发生了更改。

如果以同步方式调用 **NotifyAddrChange** 函数，则会在下次 IPv4 地址更改时发送通知，直到应用程序终止。

如果调用方指定 **句柄** 变量和 **OVERLAPPED** 结构，则 **NotifyAddrChange** 函数调用是异步的，调用方可以使用返回的 **句柄** 和 **OVERLAPPED** 结构，以使用 **GetOverlappedResult** 函数接收 IPv4 地址更改的异步通知。有关使用 **句柄** 和 **OVERLAPPED** 结构接收通知的信息，请参阅以下主题：

- [同步和重叠输入和输出](#)
- [GetOverlappedResult](#)

[CancelIPChangeNotify](#) 函数取消之前请求的 IPv4 地址和路由更改的通知，并成功调用 **NotifyAddrChange** 或 **NotifyRouteChange** 函数。

当应用程序收到更改通知后，应用程序可以调用 [GetIpAddrTable](#) 或 [GetAdaptersAddresses](#) 函数来检索 IPv4 地址表以确定更改的内容。如果通知应用程序并要求通知下一次更改，则必须再次调用 **NotifyAddrChange** 函数。

如果以异步方式调用 **NotifyAddrChange** 函数，则会在下一个 IPv4 地址更改时发送通知，直到应用程序通过调用 [CancelIPChangeNotify](#) 函数取消通知或应用程序终止。如果应用程序终止，系统将自动取消通知的注册。仍建议应用程序在终止之前显式取消任何通知。

在系统关闭或重新启动时，通知的任何注册都不会保留。

在 Windows Vista 及更高版本上，[NotifyInterfaceChange](#) 函数可用于注册以获取本地计算机上的 IPv4 和 IPv6 接口更改的通知。

示例

以下示例等待将 IP 地址映射到接口的表中发生更改。

C++

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <windows.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

void main()
{
    OVERLAPPED overlap;
    DWORD ret;

    HANDLE hand = NULL;
    overlap.hEvent = WSACreateEvent();

    ret = NotifyAddrChange(&hand, &overlap);

    if (ret != NO_ERROR)
    {
        if (WSAGetLastError() != WSA_IO_PENDING)
        {
            printf("NotifyAddrChange error...%d\n", WSAGetLastError());
            return;
        }
    }

    if ( WaitForSingleObject(overlap.hEvent, INFINITE) == WAIT_OBJECT_0 )
        printf("IP Address table changed..\n");
}
```

要求

最低受支持的客户端

Windows 2000 Professional [仅限桌面应用]

最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CancelIPChangeNotify](#)

[GetAdaptersAddresses](#)

[GetIpAddrTable](#)

[GetOverlappedResult](#)

[IP 帮助程序函数参考](#)

[NotifyIpInterfaceChange](#)

[NotifyRouteChange](#)

[OVERLAPPED](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

NotifyIpInterfaceChange 函数 (netioapi.h)

项目2023/08/25

NotifyIpInterfaceChange 函数注册本地计算机上所有 IP 接口、IPv4 接口或 IPv6 接口的更改时会收到通知。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
NotifyIpInterfaceChange(
    [in]      ADDRESS_FAMILY           Family,
    [in]      PIPINTERFACE_CHANGE_CALLBACK Callback,
    [in]      PVOID                   CallerContext,
    [in]      BOOLEAN                InitialNotification,
    [in, out] HANDLE                *NotificationHandle
);
```

parameters

[in] Family

要注册更改通知的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和PF_ 协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 Ws2def.h 头文件中定义了此成员的可能值。请注意，Ws2def.h 头文件会自动包含在 Winsock2.h 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	地址系列未指定。指定此参数后，此函数将同时注册 IPv4 和 IPv6 更改通知。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数后，此函数仅注册 IPv4 更改通知。

AF_INET6

23

Internet 协议版本 6 (IPv6) 地址系列。 指定此参数时，此函数仅注册 IPv6 更改通知。

[in] Callback

指向发生更改时要调用的函数的指针。 收到接口通知时，将调用此函数。

[in] CallerContext

收到接口通知时传递给 *Callback* 参数中指定的 *回调* 函数的用户上下文。

[in] InitialNotification

一个值，该值指示是否应在更改通知注册完成后立即调用回调。 此初始通知并不表示 IP 接口发生了更改。 此参数用于确认回调已注册。

[in, out] NotificationHandle

用于返回句柄的指针，该句柄稍后可用于取消注册更改通知。 成功后，此参数中会返回通知句柄。 如果发生错误，则返回 **NUL**。

返回值

如果函数成功，则返回值 **NO_ERROR**。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_HANDLE	遇到无效句柄时发生内部错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果 <i>Family</i> 参数不是 AF_INET 、 AF_INET6 或 AF_UNSPEC ，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存不足。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

NotifyIpInterfaceChange 函数在 Windows Vista 及更高版本上定义。

Family 参数必须设置为 **AF_INET**、**AF_INET6** 或 **AF_UNSPEC**。

对 *Callback* 参数中指定的回调函数的调用进行序列化。 回调函数应定义为 VOID 类型的函数。 传递给回调函数的参数包括：

参数	说明
IN PVOID CallerContext	注册通知时传递给 NotifyIpInterfaceChange 函数的 <i>CallerContext</i> 参数。
IN PMIB_IPINTERFACE_ROW 行可选	指向已更改的接口 的 MIB_IPINTERFACE_ROW 项 的指针。 将 <i>NotificationType</i> 参数中传递给回调函数的 MIB_NOTIFICATION_TYPE 值设置为 MibInitialNotification 时，此参数是 NULL 指针。 只有在注册通知时，传递给 NotifyIpInterfaceChange 的 <i>InitialNotification</i> 参数设置为 TRUE 时，才会发生这种情况。
IN MIB_NOTIFICATION_TYPE NotificationType	通知类型。 此成员可以是 <i>Netioapi.h</i> 头文件中定义的 MIB_NOTIFICATION_TYPE 枚举类型的值之一。

Callback 参数中指定的回调函数必须在调用 **NotifyIpInterfaceChange** 函数的应用程序所在的进程中实现。 如果回调函数位于单独的 DLL 中，则应先加载 DLL，然后再调用 **NotifyIpInterfaceChange** 函数来注册更改通知。

当发生更改且 *Row* 参数不为 NULL 时收到回调函数时，指向 *Row* 参数中传递的 [MIB_IPINTERFACE_ROW](#) 结构的指针包含不完整的数据。 **MIB_IPINTERFACE_ROW** 结构中返回的信息只是应用程序调用 [GetIpInterfaceEntry](#) 函数来查询更改的 IP 接口上的完整信息。 收到回调函数时，应用程序应分配 **MIB_IPINTERFACE_ROW** 结构，并使用接收的 *Row* 参数指向的 [MIB_IPINTERFACE_ROW](#) 结构中的 **Family**、**InterfaceLuid** 和 **InterfaceIndex** 成员对其进行初始化。 应将指向此新初始化 **MIB_IPINTERFACE_ROW** 结构的指针传递给 [GetIpInterfaceEntry](#) 函数，以检索有关已更改的 IP 接口的完整信息。

回调指示中使用的 *Row* 参数指向的内存由操作系统管理。 接收通知的应用程序绝不应尝试释放 *Row* 参数指向的内存。

若要取消注册更改通知，请调用 [CancelMibChangeNotify2](#) 函数，传递 **NotifyIpInterfaceChange** 返回的 *NotificationHandle* 参数。

应用程序无法从当前正在为同一 *NotificationHandle* 参数执行通知回调函数的线程的上下文调用 [CancelMibChangeNotify2](#) 函数。 否则，执行该回调的线程将导致死锁。 因此，不能在通知回调例程中直接调用 [CancelMibChangeNotify2](#) 函数。 在更一般的情况下，执行 [CancelMibChangeNotify2](#) 函数的线程不能拥有执行通知回调操作的线程将等待的资源，因为这将导致类似的死锁。 应从其他线程调用 [CancelMibChangeNotify2](#) 函数，接收通知回调的线程不依赖于该线程。

调用 `NotifyIpInterfaceChange` 函数注册更改通知后，将继续发送这些通知，直到应用程序取消更改通知的注册或应用程序终止。如果应用程序终止，系统将自动取消注册更改通知的任何注册。仍建议应用程序在终止之前显式取消注册更改通知。

在系统关闭或重新启动时，更改通知的任何注册都不会保留。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CancelMibChangeNotify2](#)

[GetIfEntry2](#)

[GetIfStackTable](#)

[GetIfTable2](#)

[GetInvertedIfStackTable](#)

[GetIpInterfaceEntry](#)

[IP 帮助程序函数参考](#)

[InitializeIpInterfaceEntry](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

[MIB_IPINTERFACE_ROW](#)

[MIB_NOTIFICATION_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

notifyNetworkConnectivityHintChange 函数 (netioapi.h)

项目2023/08/25

注册应用程序定义的回调函数，该函数将在聚合网络连接级别和成本提示更改时调用。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
NotifyNetworkConnectivityHintChange(
    [in]    PNETWORK_CONNECTIVITY_HINT_CHANGE_CALLBACK Callback,
    [in]    PVOID                         CallerContext,
    [in]    BOOLEAN                        InitialNotification,
    [out]   PHANDLE                       NotificationHandle
);
```

parameters

[in] Callback

PNETWORK_CONNECTIVITY_HINT_CHANGE_CALLBACK 类型的函数指针，指向应用程序定义的回调函数。当发生网络连接级别或成本更改时，将调用回调函数。

[in] CallerContext

特定于用户的调用方上下文。此上下文将提供给回调函数。

[in] InitialNotification

True 如果应提供初始化通知，则为 ;否则为 false。

[out] NotificationHandle

指向 HANDLE 的指针。函数将 值设置为通知注册的句柄。

返回值

如果函数成功，则返回值 NO_ERROR。否则，将返回错误代码。

注解

若要取消注册更改通知，请调用 `CancelMibChangeNotify2` 函数，传递 `NotifyNetworkConnectivityHintChange` 返回的 `NotificationHandle` 参数。

要求

最低受支持的客户端	Windows 10, 版本 2004 (10.0;内部版本 19041)
最低受支持的服务器	Windows Server 版本 2004 (10.0;内部版本 19041)
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

- [PNETWORK_CONNECTIVITY_HINT_CHANGE_CALLBACK](#)
- [CancelMibChangeNotify2](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

notifyRouteChange 函数 (iphlpapi.h)

项目2023/08/24

只要 IPv4 路由表中发生更改， NotifyRouteChange 函数就会向调用方发送通知。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD NotifyRouteChange(
    [out] PHANDLE     Handle,
    [in]  LPOVERLAPPED overlapped
);
```

parameters

[out] Handle

指向 HANDLE 变量的指针，该变量接收要用于异步通知的句柄。

[in] overlapped

指向 OVERLAPPED 结构的指针，该结构通知调用方路由表中的任何更改。

返回值

如果函数成功，则如果调用方为 *Handle* 和 *重叠*参数指定 NULL，则返回值NO_ERROR。如果调用方指定非 NULL 参数，则成功返回值ERROR_IO_PENDING。如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

返回代码	说明
ERROR_CANCELLED	正在取消注册上下文，因此调用已立即取消。
ERROR_INVALID_PARAMETER	传递的参数无效。如果 <i>Handle</i> 和 <i>重叠</i> 参数都不是 NULL，但调用进程无法写入输入参数指定的内存，则返回此错误。如果客户端已发出更改通知请求，则也会返回此错误，因此此重复请求将失败。
ERROR_NOT_ENOUGH_MEMORY	可用内存不足，无法完成该操作。
ERROR_NOT_SUPPORTED	此错误在不支持此函数的 Windows 版本（如 Windows 98/95 和 Windows NT 4.0）上返回。

注解

The

可以通过两种方式调用 **NotifyRouteChange** 函数：

- 同步方法
- 异步方法

如果调用方为 *Handle* 和 *重叠*参数指定 **NULL**，则对 **NotifyRouteChange** 的调用是同步的，并且将阻塞，直到发生 IPv4 路由表更改。在这种情况下，如果发生更改，则 **NotifyRouteChange** 函数完成以指示已发生更改。

如果以同步方式调用 **NotifyRouteChange** 函数，则会在下次 IPv4 路由更改时发送通知，直到应用程序终止。

如果调用方指定句柄变量和 **OVERLAPPED** 结构，则调用方可以使用返回的句柄和 **OVERLAPPED** 结构来接收有关 IPv4 路由表更改的异步通知。有关使用句柄和 **OVERLAPPED** 结构接收通知的信息，请参阅以下主题：

- [同步和重叠输入和输出](#)
- [GetQueuedCompletionStatus](#)
- [I/O 完成端口](#)

如果应用程序收到通知并要求通知进行下一次更改，则必须再次调用 **NotifyRouteChange** 函数。

[CancelIPChangeNotify](#) 函数通过成功调用 [NotifyAddrChange](#) 或 [NotifyRouteChange](#) 函数，取消之前请求的 IP 地址和路由更改的通知。

通知应用程序更改后，应用程序可以调用 [GetIpForwardTable](#) 或 [GetIpForwardTable2](#) 函数来检索 IPv4 路由表以确定更改的内容。如果应用程序收到通知，并且需要通知进行下一次更改，则必须再次调用 **NotifyRouteChange** 函数。

如果 **异步调用** **NotifyRouteChange** 函数，则会在下一次 IPv4 路由更改时发送通知，直到应用程序通过调用 [CancelIPChangeNotify](#) 函数取消通知或应用程序终止。如果应用程序终止，系统将自动取消通知的注册。仍建议应用程序在终止之前显式取消任何通知。

通知的任何注册不会在系统关闭或重新启动时保留。

在 Windows Vista 及更高版本上，可以使用 [NotifyRouteChange2](#) 函数注册，以便收到本地计算机上 IPv6 路由表更改的通知。

示例

以下示例等待 IP 路由表中发生更改。

C++

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <windows.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

void main()
{
    OVERLAPPED overlap;
    DWORD ret;

    HANDLE hand = NULL;
    overlap.hEvent = WSACreateEvent();

    ret = NotifyRouteChange(&hand, &overlap);

    if (ret != NO_ERROR)
    {
        if (WSAGetLastError() != WSA_IO_PENDING)
        {
            printf("NotifyRouteChange error...%d\n", WSAGetLastError());
            return;
        }
    }

    if ( WaitForSingleObject(overlap.hEvent, INFINITE) == WAIT_OBJECT_0 )
        printf("Routing table changed..\n");
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib

DLL	Iphlpapi.dll
-----	--------------

另请参阅

[CancelIPChangeNotify](#)

[GetIpForwardTable](#)

[GetIpForwardTable2](#)

[GetOverlappedResult](#)

[IP 帮助程序函数参考](#)

[NotifyAddrChange](#)

[OVERLAPPED](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

NotifyRouteChange2 函数 (netioapi.h)

项目2023/08/25

NotifyRouteChange2 函数注册本地计算机上的 IP 路由条目更改时会收到通知。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API NotifyRouteChange2(
    [in]      ADDRESS_FAMILY             AddressFamily,
    [in]      PIPFORWARD_CHANGE_CALLBACK Callback,
    [in]      PVOID                    CallerContext,
    [in]      BOOLEAN                  InitialNotification,
    [in, out] HANDLE                 *NotificationHandle
);
```

parameters

[in] AddressFamily

要注册更改通知的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和PF_ 协议系列常量的值 (相同，例如，AF_INET 和 PF_INET)，因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 Ws2def.h 头文件中定义了此成员的可能值。请注意，Ws2def.h 头文件会自动包含在 Winsock2.h 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_INET	仅注册 IPv4 路由更改通知。
AF_INET6	仅注册 IPv6 路由更改通知。
AF_UNSPEC	注册 IPv4 和 IPv6 路由更改通知。

[in] Callback

指向发生更改时要调用的函数的指针。收到 IP 路由通知时，将调用此函数。

[in] CallerContext

收到 IP 路由通知时传递给 Callback 参数中指定的 回调 函数的用户上下文。

[in] InitialNotification

一个值，该值指示是否应在更改通知注册完成后立即调用回调。此初始通知并不表示 IP 路由条目发生了更改。此参数用于确认回调已注册。

[in, out] NotificationHandle

用于返回句柄的指针，该句柄稍后可用于取消注册更改通知。成功后，此参数中会返回通知句柄。如果发生错误，则返回 NULL。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_HANDLE	遇到无效句柄时发生内部错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 Family 参数不是 AF_INET、AF_INET6 或 AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存不足。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

NotifyRouteChange2 函数在 Windows Vista 及更高版本上定义。

Family 参数必须设置为 AF_INET、AF_INET6 或 AF_UNSPEC。

对 Callback 参数中指定的回调函数的调用进行序列化。回调函数应定义为 VOID 类型的函数。传递给回调函数的参数包括：

参数	说明
IN PVOID CallerContext	注册通知时传递给 NotifyRouteChange2 函数的 CallerContext 参数。

IN PMIB_IPFORWARD_ROW2 行可选	指向已更改的 IP 路由条目 的 MIB_IPFORWARD_ROW2 项的指针。将 <i>NotificationType</i> 参数中传递给回调函数的 MIB_NOTIFICATION_TYPE 值设置为 MibInitialNotification 时，此参数是 NULL 指针。只有在注册通知时，传递给 NotifyRouteChange2 的 InitialNotification 参数设置为 TRUE 时，才会发生这种情况。
IN MIB_NOTIFICATION_TYPE NotificationType	通知类型。此成员可以是 Netioapi.h 头文件中定义的 MIB_NOTIFICATION_TYPE 枚举类型的值之一。

Callback 参数中指定的回调函数必须在调用 **NotifyRouteChange2** 函数的应用程序所在的进程中实现。如果回调函数位于单独的 DLL 中，则应先加载 DLL，然后再调用 **NotifyRouteChange2** 函数来注册更改通知。

当发生更改且 *Row* 参数不为 NULL 时收到回调函数时，指向 *Row* 参数中传递的 MIB_IPFORWARD_ROW2 结构的指针包含不完整的数据。MIB_IPFORWARD_ROW2 结构中返回的信息仅是应用程序可以调用 **GetIpForwardEntry2** 函数来查询更改的 IP 路由的完整信息。收到回调函数时，应用程序应分配 MIB_IPFORWARD_ROW2 结构，并使用接收的 *Row* 参数指向的 MIB_IPFORWARD_ROW2 结构中的 DestinationPrefix、NextHop、InterfaceLuid 和 InterfaceIndex 成员对其进行初始化。应将指向此新初始化 MIB_IPFORWARD_ROW2 结构的指针传递给 **GetIpForwardEntry2** 函数，以检索有关已更改的 IP 路由的完整信息。

回调指示中使用的 *Row* 参数指向的内存由操作系统管理。接收通知的应用程序绝不应尝试释放 *Row* 参数指向的内存。

调用 **NotifyRouteChange2** 函数注册更改通知后，将继续发送这些通知，直到应用程序取消更改通知的注册或应用程序终止。如果应用程序终止，系统将自动取消注册更改通知的任何注册。仍建议应用程序在终止之前显式取消注册更改通知。

如果系统关闭或重新启动，则更改通知的任何注册都不会保留。

若要取消更改通知的注册，请调用 **CancelMibChangeNotify2** 函数，传递 **NotifyRouteChange2** 返回的 *NotificationHandle* 参数。

应用程序无法从当前正在为同一 *NotificationHandle* 参数执行通知回调函数的线程的上下文调用 **CancelMibChangeNotify2** 函数。否则，执行该回调的线程将导致死锁。因此，不能在通知回调例程中直接调用 **CancelMibChangeNotify2** 函数。在更一般的情况下，执行 **CancelMibChangeNotify2** 函数的线程不能拥有执行通知回调操作的线程将等待的资源，因为这将导致类似的死锁。应从其他线程调用 **CancelMibChangeNotify2** 函数，接收通知回调的线程不依赖于该线程。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CancelMibChangeNotify2](#)

[CreateIpForwardEntry2](#)

[DeleteIpForwardEntry2](#)

[GetBestRoute2](#)

[GetIpForwardEntry2](#)

[GetIpForwardTable2](#)

[InitializeIpForwardEntry](#)

[MIB_IPFORWARD_ROW2](#)

[MIB_IPFORWARD_TABLE2](#)

[MIB_NOTIFICATION_TYPE](#)

[SetIpForwardEntry2](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

notifyStableUnicastIpAddressTable 函数 (netioapi.h)

项目2023/08/25

NotifyStableUnicastIpAddressTable 函数检索本地计算机上的稳定单播 IP 地址表。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
NotifyStableUnicastIpAddressTable(
    [in]      ADDRESS_FAMILY                  Family,
    [in, out] PMIB_UNICASTIPADDRESS_TABLE    *Table,
    [in]      PSTABLE_UNICAST_IPADDRESS_TABLE_CALLBACK CallerCallback,
    [in]      PVOID                           CallerContext,
    [in, out] HANDLE                         *NotificationHandle
);
```

parameters

[in] Family

要检索的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_地址系列和PF_协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织已更改，并且 Ws2def.h 头文件中定义了此成员的可能值。请注意，Ws2def.h 头文件自动包含在 Winsock2.h 中，不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_UNSPEC 0	未指定地址系列。指定此参数后，函数将检索包含 IPv4 和 IPv6 条目的稳定单播 IP 地址表。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数后，函数将检索仅包含 IPv4 条目的稳定单播 IP 地址表。

AF_INET6

23

Internet 协议版本 6 (IPv6) 地址系列。指定此参数后，函数将检索仅包含 IPv6 条目的稳定单播 IP 地址表。

[in, out] Table

指向 [MIB_UNICASTIPADDRESS_TABLE](#) 结构的指针。如果 `NotifyStableUnicastIpAddressTable` 成功，此参数将返回本地计算机上的稳定单播 IP 地址表。

当 `NotifyStableUnicastIpAddressTable` 返回 `ERROR_IO_PENDING` 指示 I/O 请求挂起时，稳定的单播 IP 地址表将返回到 `CallerCallback` 参数中的函数。

[in] CallerCallback

指向要通过稳定的单播 IP 地址表调用的函数的指针。如果 `NotifyStableUnicastIpAddressTable` 返回 `ERROR_IO_PENDING`，指示 I/O 请求处于挂起状态，则将调用此函数。

[in] CallerContext

当稳定的单播 IP 地址表 `si` 可用时，传递给 `CallerCallback` 参数中指定的回调函数的用户上下文。

[in, out] NotificationHandle

用于返回句柄的指针，该句柄可用于取消请求以检索稳定的单播 IP 地址表。如果来自 `NotifyStableUnicastIpAddressTable` 的返回值 `ERROR_IO_PENDING` 指示 I/O 请求处于挂起状态，则返回此参数。

返回值

如果函数立即成功，则返回值 `NO_ERROR`，并在 `Table` 参数中返回稳定的单播 IP 表。

如果 I/O 请求处于挂起状态，则函数返回 `ERROR_IO_PENDING` 并且当 I/O 请求使用稳定的单播 IP 地址表完成时，将调用 `CallerCallback` 参数指向的函数。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
<code>ERROR_INVALID_HANDLE</code>	遇到无效句柄时发生内部错误。
<code>ERROR_INVALID_PARAMETER</code>	向该函数传递了无效参数。如果 <code>Table</code> 参数为 <code>NULL</code> 指针， <code>NotificationHandle</code> 参数为 <code>NULL</code> 指针，或者 <code>Family</code>

	参数不是 AF_INET、AF_INET6 或 AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存不足。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

NotifyStableUnicastIpAddressTable 函数在 Windows Vista 及更高版本上定义。

如果 **NotifyStableUnicastIpAddressTable** 函数立即成功，则返回值 NO_ERROR 并且稳定的单播 IP 表在 *Table* 参数中返回。当不再需要 MIB_UNICASTIPADDRESS_TABLE 信息时，调用应用程序应使用 [FreeMibTable](#) 函数释放 *Table* 参数指向的内存。

所有单播 IP 地址（按需拨号地址除外）仅当它们处于首选状态时才会被视为稳定。对于普通单播 IP 地址条目，这将对应于设置为 **IpDadStatePreferred** 的 IP 地址 [MIB_UNICASTIPADDRESS_ROW](#) 的 **DadState** 成员。每个按需拨号地址定义其自己的稳定性指标。目前，此函数考虑的唯一按需拨号地址是本地计算机上的 Teredo 客户端使用的单播 IP 地址。

Family 参数必须设置为 AF_INET、AF_INET6 或 AF_UNSPEC。

如果 **NotifyStableUnicastIpAddressTable** 成功并返回 NO_ERROR，则 *Table* 参数返回本地计算机上的稳定单播 IP 地址表。

当 **NotifyStableUnicastIpAddressTable** 返回 ERROR_IO_PENDING 指示 I/O 请求挂起时，稳定的单播 IP 地址表将返回到 *CallerCallback* 参数中的函数。

NotifyStableUnicastIpAddressTable 函数主要由使用 Teredo 客户端的应用程序使用。

如果 Teredo 使用的单播 IP 地址在本地计算机上可用，但不在本地计算机上处于稳定（限定）状态，则 **NotifyStableUnicastIpAddressTable** 将返回 ERROR_IO_PENDING，并且最终将通过调用 *CallerCallback* 参数中的函数返回稳定的单播 IP 地址表。如果 Teredo 地址不可用或处于稳定状态，而其他单播 IP 地址处于稳定状态，则永远不会调用 *CallerCallback* 参数中的函数。

CallerCallback 参数中指定的回调函数应定义为 VOID 类型的函数。传递给回调函数的参数包括：

参数	说明
IN PVOID CallerContext	注册通知时传递给 NotifyStableUnicastIpAddressTable 函数的 <i>CallerContext</i> 参数。

IN PMIB_UNICASTIPADDRESS_TABLE AddressTable	指向包含本地计算机上稳定单播 IP 地址表的 MIB_UNICASTIPADDRESS_TABLE 的指针。
---	--

CallerCallback 参数中指定的回调函数必须在调用 **NotifyStableUnicastIpAddressTable** 函数的应用程序所在同一进程中实现。如果回调函数位于单独的 DLL 中，则应先加载 DLL，然后再调用 **NotifyStableUnicastIpAddressTable** 函数来注册更改通知。

回调指示中使用的 *AddressTable* 参数指向的内存由操作系统分配。当不再需要 MIB_UNICASTIPADDRESS_TABLE 信息时，接收通知的应用程序应使用 **FreeMibTable** 函数释放 *AddressTable* 参数指向的内存。

调用 **NotifyStableUnicastIpAddressTable** 函数注册更改通知后，将继续发送这些通知，直到应用程序取消注册更改通知或应用程序终止为止。如果应用程序终止，系统将自动取消注册更改通知的任何注册。仍建议应用程序在终止之前显式取消注册任何更改通知。

如果系统关闭或重新启动，则更改通知的任何注册都不会保留。

若要取消注册更改通知，请调用 **CancelMibChangeNotify2** 函数，传递由 **NotifyStableUnicastIpAddressTable** 返回的 *NotificationHandle* 参数。

应用程序无法从当前正在为同一 *NotificationHandle* 参数执行通知回调函数的线程上下文调用 **CancelMibChangeNotify2** 函数。否则，执行该回调的线程将导致死锁。因此，不能在通知回调例程中直接调用 **CancelMibChangeNotify2** 函数。在更一般的情况下，执行 **CancelMibChangeNotify2** 函数的线程不能拥有执行通知回调操作的线程将等待的资源，因为这会导致类似的死锁。**CancelMibChangeNotify2** 函数应从另一个线程调用，接收通知回调的线程不依赖于该线程。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CancelMibChangeNotify2](#)

[CreateUnicastIpAddressEntry](#)

[DeleteUnicastIpAddressEntry](#)

[FreeMibTable](#)

[GetTeredoPort](#)

[GetUnicastIpAddressEntry](#)

[GetUnicastIpAddressTable](#)

[IP 帮助程序函数参考](#)

[InitializeUnicastIpAddressEntry](#)

[MIB_NOTIFICATION_TYPE](#)

[MIB_UNICASTIPADDRESS_ROW](#)

[MIB_UNICASTIPADDRESS_TABLE](#)

[NotifyTeredoPortChange](#)

[NotifyUnicastIpAddressChange](#)

[SetUnicastIpAddressEntry](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

notifyTeredoPortChange 函数 (netioapi.h)

项目2023/08/25

NotifyTeredoPortChange 函数注册，以便在本地计算机上 Teredo 客户端用于 Teredo 服务端口的 UDP 端口号发生更改时收到通知。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API NotifyTeredoPortChange(
    [in]      PTEREDO_PORT_CHANGE_CALLBACK Callback,
    [in]      PVOID             CallerContext,
    [in]      BOOLEAN           InitialNotification,
    [in, out] HANDLE          *NotificationHandle
);
```

parameters

[in] Callback

指向在 Teredo 客户端端口发生更改时要调用的函数的指针。 收到 Teredo 端口更改通知时，将调用此函数。

[in] CallerContext

收到 Teredo 端口更改通知时传递给 *Callback* 参数中指定的回调函数的用户上下文。

[in] InitialNotification

一个值，该值指示是否应在注册完成更改通知后立即调用回调。 此初始通知并不指示 Teredo 客户端端口发生了更改。 此参数的用途，用于确认回调已注册。

[in, out] NotificationHandle

用于返回句柄的指针，该句柄稍后可用于取消注册更改通知。 成功后，此参数中将返回通知句柄。 如果发生错误，则返回 NULL。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_HANDLE	遇到无效句柄时发生内部错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>Callback</i> 参数为 NULL 指针，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存不足。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

NotifyTeredoPortChange 函数在 Windows Vista 及更高版本上定义。

GetTeredoPort 函数可用于检索 Teredo 客户端用于 Teredo 服务端口的初始 UDP 端口号。

Teredo 端口是动态的，可以在本地计算机上重启 Teredo 客户端时随时更改。应用程序可以通过调用 **NotifyTeredoPortChange** 函数注册，以便在 Teredo 服务端口更改时收到通知。

对 *Callback* 参数中指定的回调函数的调用进行序列化。回调函数应定义为 VOID 类型的函数。传递给回调函数的参数包括：

参数	说明
IN PVOID CallerContext	注册通知时传递给 NotifyTeredoPortChange 函数的 <i>CallerContext</i> 参数。
IN USHORT Port	Teredo 客户端当前使用的 UDP 端口号。将 <i>NotificationType</i> 参数中传递给回调函数的 MIB_NOTIFICATION_TYPE 值设置为 MibInitialNotification 时，此参数为零。只有在注册通知时传递给 NotifyTeredoPortChange 的 <i>InitialNotification</i> 参数设置为 TRUE 时，才会发生这种情况。
IN MIB_NOTIFICATION_TYPE NotificationType	通知类型。此成员可以是 <i>Netioapi.h</i> 头文件中定义的 MIB_NOTIFICATION_TYPE 枚举类型的值之一。

Callback 参数中指定的回调函数必须在调用 **NotifyTeredoPortChange** 函数的应用程序所在的同一进程中实现。如果回调函数位于单独的 DLL 中，则应在调用

NotifyTeredoPortChange 函数以注册更改通知之前加载该 DLL。

调用 **NotifyTeredoPortChange** 函数注册更改通知后，将继续发送这些通知，直到应用程序取消注册更改通知或应用程序终止为止。如果应用程序终止，系统将自动取消注册更改通知的任何注册。仍建议应用程序在终止之前显式取消注册更改通知。

在系统关闭或重新启动时，更改通知的任何注册都不会保留。

若要取消注册更改通知，请调用 [CancelMibChangeNotify2](#) 函数，传递 **NotifyTeredoPortChange** 返回的 *NotificationHandle* 参数。

应用程序无法从当前正在为同一 *NotificationHandle* 参数执行通知回调函数的线程上下文调用 [CancelMibChangeNotify2](#) 函数。否则，执行该回调的线程将导致死锁。因此，不能在通知回调例程中直接调用 [CancelMibChangeNotify2](#) 函数。在更一般的情况下，执行 [CancelMibChangeNotify2](#) 函数的线程不能拥有执行通知回调操作的线程将等待的资源，因为这会导致类似的死锁。[CancelMibChangeNotify2](#) 函数应从另一个线程调用，接收通知回调的线程不依赖于该线程。

Teredo 客户端还使用静态 UDP 端口 3544 来侦听在 RFC 4380 中定义的多播 IPv4 地址 224.0.0.253 上发送的多播流量。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4380.txt>。

NotifyTeredoPortChange 函数主要由防火墙应用程序使用，以便配置相应的例外，以允许传入和传出 Teredo 流量。

[NotifyStableUnicastIpAddressTable](#) 函数主要由使用 Teredo 客户端的应用程序使用。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CancelMibChangeNotify2](#)

[GetTeredoPort](#)

[NotifyStableUnicastIpAddressTable](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

NotifyUnicastIpAddressChange 函数 (netioapi.h)

项目2023/08/25

NotifyUnicastIpAddressChange 函数注册以通知本地计算机上所有单播 IP 接口、单播 IPv4 地址或单播 IPv6 地址的更改。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
NotifyUnicastIpAddressChange(
    [in]      ADDRESS_FAMILY           Family,
    [in]      PUNICAST_IPADDRESS_CHANGE_CALLBACK Callback,
    [in]      PVOID                  CallerContext,
    [in]      BOOLEAN                InitialNotification,
    [in, out] HANDLE                *NotificationHandle
);
```

parameters

[in] Family

要注册更改通知的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和PF_ 协议系列常量的值（相同，例如，AF_INET 和 PF_INET），因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 Ws2def.h 头文件中定义了此成员的可能值。请注意，Ws2def.h 头文件会自动包含在 Winsock2.h 中，永远不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

值	含义
AF_INET	仅注册单播 IPv4 地址更改通知。
AF_INET6	仅注册单播 IPv6 地址更改通知。
AF_UNSPEC	注册单播 IPv4 和 IPv6 地址更改通知。

[in] Callback

指向发生更改时要调用的函数的指针。 收到单播 IP 地址通知时，将调用此函数。

[in] CallerContext

收到接口通知时传递给 Callback 参数中指定的 *回调* 函数的用户上下文。

[in] InitialNotification

一个值，该值指示是否应在更改通知注册完成后立即调用回调。 此初始通知并不表示单播 IP 地址发生了更改。 此参数用于确认回调已注册。

[in, out] NotificationHandle

用于返回句柄的指针，该句柄稍后可用于取消注册更改通知。 成功后，此参数中会返回通知句柄。 如果发生错误，则返回 NULL。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_HANDLE	遇到无效句柄时发生内部错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。 如果 Family 参数不是 AF_INET、AF_INET6或AF_UNSPEC，则返回此错误。
ERROR_NOT_ENOUGH_MEMORY	内存不足。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`NotifyUnicastIpAddressChange` 函数是在 Windows Vista 及更高版本上定义的。

Family 参数必须设置为 AF_INET、AF_INET6 或 AF_UNSPEC。

对 *Callback* 参数中指定的回调函数的调用进行序列化。 回调函数应定义为 VOID 类型的函数。 传递给回调函数的参数包括：

参数	说明
----	----

IN PVOID CallerContext	注册通知时传递给 <code>NotifyUnicastIpAddressChange</code> 函数的 <i>CallerContext</i> 参数。
IN PMIB_UNICASTIPADDRESS_ROW 行可选	指向已更改的单播 IP 地址 <code>MIB_UNICASTIPADDRESS_ROW</code> 条目的指针。将 <i>NotificationType</i> 参数中传递给回调函数的 <code>MIB_NOTIFICATION_TYPE</code> 值设置为 <code>MibInitialNotification</code> 时，此参数是 NULL 指针。只有在注册通知时，传递给 <code>NotifyUnicastIpAddressChange</code> 的 <i>InitialNotification</i> 参数设置为 TRUE 时，才会发生这种情况。
IN MIB_NOTIFICATION_TYPE <i>NotificationType</i>	通知类型。此成员可以是 <code>Netioapi.h</code> 头文件中定义的 <code>MIB_NOTIFICATION_TYPE</code> 枚举类型的值之一。

Callback 参数中指定的回调函数必须在与调用 `NotifyUnicastIpAddressChange` 函数的应用程序相同的进程中实现。如果回调函数位于单独的 DLL 中，则应先加载 DLL，然后再调用 `NotifyUnicastIpAddressChange` 函数来注册更改通知。

当发生更改且 *Row* 参数不为 NULL 时收到回调函数时，指向 *Row* 参数中传递的 `MIB_UNICASTIPADDRESS_ROW` 结构的指针包含不完整的数据。

`MIB_UNICASTIPADDRESS_ROW` 结构中返回的信息只是应用程序可以调用 `GetUnicastIpAddressEntry` 函数来查询更改的 IP 地址的完整信息。收到回调函数时，应用程序应分配 `MIB_UNICASTIPADDRESS_ROW` 结构，并使用接收的 *Row* 参数指向的 `MIB_UNICASTIPADDRESS_ROW` 结构中的 `Address`、`InterfaceLuid` 和 `InterfaceIndex` 成员对其进行初始化。应将指向此新初始化 `MIB_UNICASTIPADDRESS_ROW` 结构的指针传递给 `GetUnicastIpAddressEntry` 函数，以检索有关已更改的单播 IP 地址的完整信息。

回调指示中使用的 *Row* 参数指向的内存由操作系统管理。接收通知的应用程序绝不应尝试释放 *Row* 参数指向的内存。

调用 `NotifyUnicastIpAddressChange` 函数注册更改通知后，将继续发送这些通知，直到应用程序取消注册更改通知或应用程序终止。如果应用程序终止，系统将自动取消注册更改通知的任何注册。仍建议应用程序在终止之前显式取消注册更改通知。

如果系统关闭或重新启动，则更改通知的任何注册都不会保留。

若要取消注册更改通知，请调用 `CancelMibChangeNotify2` 函数，传递 `NotifyUnicastIpAddressChange` 返回的 *NotificationHandle* 参数。

应用程序无法从当前正在为同一 *NotificationHandle* 参数执行通知回调函数的线程的上下文调用 `CancelMibChangeNotify2` 函数。否则，执行该回调的线程将导致死锁。因此，不能在通知回调例程中直接调用 `CancelMibChangeNotify2` 函数。在更一般的情况下，执行 `CancelMibChangeNotify2` 函数的线程不能拥有执行通知回调操作的线程将等待的

资源，因为这将导致类似的死锁。应从其他线程调用 `CancelMibChangeNotify2` 函数，接收通知回调的线程不依赖于该线程。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	<code>netioapi.h</code> (包括 <code>Iphlpapi.h</code>)
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

请参阅

[CancelMibChangeNotify2](#)

[CreateUnicastIpAddressEntry](#)

[DeleteUnicastIpAddressEntry](#)

[GetUnicastIpAddressEntry](#)

[GetUnicastIpAddressTable](#)

[IP 帮助程序函数参考](#)

[InitializeUnicastIpAddressEntry](#)

[MIB_NOTIFICATION_TYPE](#)

[MIB_UNICASTIPADDRESS_ROW](#)

[MIB_UNICASTIPADDRESS_TABLE](#)

[NotifyStableUnicastIpAddressTable](#)

[SetUnicastIpAddressEntry](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

ParseNetworkString 函数 (iphlpapi.h)

项目2023/08/24

ParseNetworkString 函数分析输入网络字符串，并检查它是否是指定 IP 网络字符串类型的合法表示形式。如果字符串与类型及其规范匹配，函数可以选择返回分析的结果。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD ParseNetworkString(
    [in]          const WCHAR      *NetworkString,
    [in]          DWORD           Types,
    [out, optional] PNET_ADDRESS_INFO AddressInfo,
    [out, optional] USHORT          *PortNumber,
    [out, optional] BYTE            *PrefixLength
);
```

parameters

[in] NetworkString

指向要分析的以 NULL 结尾的网络字符串的指针。

[in] Types

要分析的 IP 网络字符串的类型。此参数由 *Iphlpapi.h* 头文件中定义的网络字符串类型之一组成。

值	含义
NET_STRING_IPV4_ADDRESS 0x00000001	<p><i>NetworkString</i> 参数使用 Internet 标准点十进制表示法指向 IPv4 地址。网络字符串中不可包含网络端口和前缀。</p> <p>下面是一个示例网络字符串：</p> <p>192.168.100.10</p>
NET_STRING_IPV4_SERVICE 0x00000002	<p><i>NetworkString</i> 参数使用 Internet 标准点十进制表示法指向 IPv4 服务。网络字符串中需要一个网络端口。网络字符串中不得存在前缀。</p> <p>下面是一个示例网络字符串：</p>

NET_STRING_IPV4_NETWORK 0x00000004	<i>NetworkString</i> 参数使用 Internet 标准点十进制表示法指向 IPv4 网络。需要在网络字符串中包含使用无类别域际路由 (CIDR) 表示法的网络前缀。网络字符串中不可包含网络端口。
	下面是一个示例网络字符串： 192.168.100/24
NET_STRING_IPV6_ADDRESS 0x00000008	<i>NetworkString</i> 参数使用 Internet 标准十六进制编码指向 IPv6 地址。网络字符串中可能存在 IPv6 范围 ID。网络字符串中不可包含网络端口和前缀。
	下面是一个示例网络字符串： 21DA: 00D3: 0000: 2F3B: 02AA: 00FF: FE28: 9C5A%2
NET_STRING_IPV6_ADDRESS_NO_SCOPE 0x00000008	<i>NetworkString</i> 参数使用 Internet 标准十六进制编码指向 IPv6 地址。网络字符串中不得存在 IPv6 范围 ID。网络字符串中不可包含网络端口和前缀。
	下面是一个示例网络字符串： 21DA:00D3:0000:2F3B:02AA:00FF:FE28:9C5A
NET_STRING_IPV6_SERVICE 0x00000020	<i>NetworkString</i> 参数使用 Internet 标准十六进制编码指向 IPv6 服务。网络字符串中需要一个网络端口。网络字符串中可能存在 IPv6 范围 ID。网络字符串中不得存在前缀。
	具有范围 ID 的示例网络字符串如下： [21DA: 00D3: 0000: 2F3B: 02AA: 00FF: FE28: 9C5A%2]: 8080
NET_STRING_IPV6_SERVICE_NO_SCOPE 0x00000040	<i>NetworkString</i> 参数使用 Internet 标准十六进制编码指向 IPv6 服务。网络字符串中需要一个网络端口。网络字符串中不得存在 IPv6 范围 ID。网络字符串中不得存在前缀。
	下面是一个示例网络字符串： 21DA: 00D3: 0000: 2F3B: 02AA: 00FF: FE28: 9C5A: 8080
NET_STRING_IPV6_NETWORK 0x00000080	<i>NetworkString</i> 参数使用 Internet 标准十六进制编码指向 IPv6 网络。网络字符串中需包含 CIDR 表示法中的网络前缀。网络字符串中不可包含网络端口和范围 ID。

下面是一个示例网络字符串：

21DA: D3: : /48

NET_STRING_NAMED_ADDRESS
0x00000100

NetworkString 参数使用域名系统 (DNS) 名称指向 Internet 地址。 网络字符串中不可包含网络端口和前缀。

下面是一个示例网络字符串：

www.microsoft.com

NET_STRING_NAMED_SERVICE
0x00000200

NetworkString 参数使用 DNS 名称指向 Internet 服务。 网络字符串中必须存在网络端口。

下面是一个示例网络字符串：

www.microsoft.com:80

NET_STRING_IP_ADDRESS
0x00000009

NetworkString 参数使用 Internet 标准点十进制表示法指向 IPv4 地址或使用 Internet 标准十六进制编码的 IPv6 地址。 网络字符串中可能存在 IPv6 范围 ID。 网络字符串中不可包含网络端口和前缀。

此类型与 **NET_STRING_IPV4_ADDRESS** 或 **NET_STRING_IPV6_ADDRESS** 类型匹配。

NET_STRING_IP_ADDRESS_NO_SCOPE
0x00000011

NetworkString 参数使用 Internet 标准点十进制表示法指向 IPv4 地址或使用 Internet 标准十六进制编码的 IPv6 地址。 网络字符串中不得存在 IPv6 范围 ID。 网络字符串中不可包含网络端口和前缀。

此类型与 **NET_STRING_IPV4_ADDRESS** 或 **NET_STRING_IPV6_ADDRESS_NO_SCOPE** 类型匹配。

NET_STRING_IP_SERVICE
0x00000022

NetworkString 参数指向 IPv4 服务或 IPv6 服务。 网络字符串中需要一个网络端口。 网络字符串中可能存在 IPv6 范围 ID。 网络字符串中不得存在前缀。

此类型匹配 **NET_STRING_IPV4_SERVICE** 或 **NET_STRING_IPV6_SERVICE** 类型。

NET_STRING_IP_SERVICE_NO_SCOPE
0x00000042

NetworkString 参数指向 IPv4 服务或 IPv6 服务。 网络字符串中需要一个网络端口。 网络字符串中不得存在 IPv6 范围 ID。 网络字符串中不得存在前缀。

此类型匹配 **NET_STRING_IPV4_SERVICE** 或 **NET_STRING_IPV6_SERVICE_NO_SCOPE** 类型。

NET_STRING_IP_NETWORK
0x00000084

NetworkString 参数指向 IPv4 或 IPv6 网络。 网络字符串中需包含 CIDR 表示法中的网络前缀。 网络中不得存在网络端口或范围 ID。

	此类型匹配 NET_STRING_IPV4_NETWORK 或 NET_STRING_IPV6_NETWORK 类型。
NET_STRING_ANY_ADDRESS 0x00000209	<i>NetworkString</i> 参数指向 Internet 标准点十进制表示法中的 IPv4 地址、Internet 标准十六进制编码中的 IPv6 地址或 DNS 名称。IPv6 地址的网络字符串中可能存在 IPv6 范围 ID。网络字符串中不可包含网络端口和前缀。 此类型与 NET_STRING_NAMED_ADDRESS 或 NET_STRING_IP_ADDRESS 类型匹配。
NET_STRING_ANY_ADDRESS_NO_SCOPE 0x00000211	<i>NetworkString</i> 参数指向 Internet 标准点十进制表示法中的 IPv4 地址、Internet 标准十六进制编码中的 IPv6 地址或 DNS 名称。IPv6 地址的网络字符串中不得存在 IPv6 范围 ID。网络字符串中不可包含网络端口和前缀。 此类型匹配 NET_STRING_NAMED_ADDRESS 或 NET_STRING_IP_ADDRESS_NO_SCOPE 类型。
NET_STRING_ANY_SERVICE 0x00000222	<i>NetworkString</i> 参数使用 IP 地址表示法或 DNS 名称指向 IPv4 服务或 IPv6 服务。网络字符串中需要一个网络端口。网络字符串中可能存在 IPv6 范围 ID。网络字符串中不得存在前缀。 此类型匹配 NET_STRING_NAMED_SERVICE 或 NET_STRING_IP_SERVICE 类型。
NET_STRING_ANY_SERVICE_NO_SCOPE 0x00000242	<i>NetworkString</i> 参数使用 IP 地址表示法或 DNS 名称指向 IPv4 服务或 IPv6 服务。网络字符串中需要一个网络端口。网络字符串中不得存在 IPv6 范围 ID。网络字符串中不得存在前缀。 此类型匹配 NET_STRING_NAMED_SERVICE 或 NET_STRING_IP_SERVICE_NO_SCOPE 类型。

[out, optional] AddressInfo

成功后，如果此参数中未传递 NULL 指针，函数将返回指向包含已分析 IP 地址信息的 NET_ADDRESS_INFO 结构的指针。

[out, optional] PortNumber

成功后，如果此参数中未传递 NULL 指针，则函数将返回一个指针，该指针将按主机顺序返回指向已分析的网络端口。如果 *NetworkString* 参数中不存在网络端口，则返回指向零值的指针。

[out, optional] PrefixLength

成功后，如果此参数中未传递 `NULL` 指针，函数将返回指向已分析前缀长度的指针。如果 `NetworkString` 参数中不存在前缀，则返回指向值 `-1` 的指针。

返回值

如果函数成功，则返回值为 `ERROR_SUCCESS`。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
<code>ERROR_INSUFFICIENT_BUFFER</code>	传递给函数的缓冲区太小。如果 <code>AddressInfo</code> 参数指向的缓冲区太小而无法保存分析的网络地址，则返回此错误。
<code>ERROR_INVALID_PARAMETER</code>	向该函数传递了无效参数。如果在 <code>NetworkString</code> 参数中传递 <code>NULL</code> 指针，则返回此错误

注解

`ParseNetworkString` 函数分析在 `NetworkString` 参数中传递的输入网络字符串，并检查它是否是 `Types` 参数中指定的其中一种字符串类型的合法表示形式。如果字符串与类型及其规范匹配，则该函数会成功，并且当这些参数不是 `NULL` 指针时，可以选择将分析的结果返回给可选 `AddressInfo`、`PortNumber` 和 `PrefixLength` 参数中的调用方。

`ParseNetworkString` 函数可以使用 DNS 名称分析 IPv4 或 IPv6 地址、服务和网络的表示形式，以及命名的 Internet 地址和服务。

`AddressInfo` 参数指向的 [NET_ADDRESS_INFO](#) 结构。 `SOCKADDR_IN` 和 `SOCKADDR` 结构在 `Winsock2.h` 头文件自动包含的 `Ws2def.h` 头文件中定义。`SOCKADDR_IN6` 结构在 `Ws2ipdef.h` 头文件中定义，该文件由 `Ws2tcpip.h` 头文件自动包含。若要使用 `ParseNetworkString` 函数和 `NET_ADDRESS_INFO` 结构，`Winsock2.h` 和 `Ws2tcpip.h` 头文件必须包含在 `Iphlpapi.h` 头文件之前。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows

标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[NET_ADDRESS_FORMAT](#)

[NET_ADDRESS_INFO](#)

[SOCKADDR](#)

[SOCKADDR_IN](#)

[SOCKADDR_IN6](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

RegisterInterfaceTimestampConfigChange 函数 (iphlpapi.h)

项目2023/08/24

注册用户实现的回调函数，系统调用该函数以通知时间戳功能更改。可以通过调用 [UnregisterInterfaceTimestampConfigChange](#) 来取消注册。

有关详细信息和代码示例，请参阅[数据包时间戳](#)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD RegisterInterfaceTimestampConfigChange(
    PINTERFACE_TIMESTAMP_CONFIG_CHANGE_CALLBACK Callback,
    PVOID CallerContext,
    HIFTIMESTAMPCHANGE *NotificationHandle
);
```

parameters

Callback

类型: `_In_ PINTERFACE_TIMESTAMP_CONFIG_CHANGE_CALLBACK`

回调函数，在时间戳功能发生更改时调用。

CallerContext

类型: `_In_opt_ PVOID`

调用方分配的可选上下文。

NotificationHandle

类型: `_Out_ HIFTIMESTAMPCHANGE`

由函数返回的句柄，用于标识注册。

返回值

类型: `DWORD`

指示成功或失败的 DWORD 返回代码。

要求

最低受支持的客户端	Windows 10内部版本 20348
最低受支持的服务器	Windows 10内部版本 20348
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

- [包时间戳](#)
- [UnregisterInterfaceTimestampConfigChange](#)

反馈

此页面是否有帮助？



[在 Microsoft Q&A 获得帮助](#)

ResolveIpNetEntry2 函数 (netioapi.h)

项目2023/08/25

ResolveIpNetEntry2 函数解析本地计算机上邻居 IP 地址条目的物理地址。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API ResolveIpNetEntry2(
    [in, out]      PMIB_IPNET_ROW2     Row,
    [in, optional] const SOCKADDR_INET *SourceAddress
);
```

parameters

[in, out] Row

指向邻居 IP 地址条目 MIB_IPNET_ROW2 结构条目的指针。成功返回后，将使用邻居 IP 地址的属性更新此结构。

[in, optional] SourceAddress

指向可选源 IP 地址的指针，用于选择要为其发送邻居 IP 地址条目的请求的接口。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_BAD_NET_NAME	找不到网络名称。如果无法访问具有邻居 IP 地址的网络，则返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数指向 MIB_IPNET_ROW2 的 Address 成员未设置为有效的 IPv4 或 IPv6 地址，或者 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员均未指定，则返回此错误。如果在 Address 成员中传递了环回地址，也会返回此错误。

ERROR_NOT_FOUND	找不到指定的接口。如果找不到 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Row 参数指向的 MIB_IPNET_ROW2 的 Address 成员中指定了 IPv4 地址，或者本地计算机上没有 IPv6 堆栈，并且 Address 成员中指定了 IPv6 地址，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`ResolveIpNetEntry2` 函数在 Windows Vista 及更高版本上定义。

`ResolveIpNetEntry2` 函数用于解析本地计算机上邻居 IP 地址条目的物理地址。此函数刷新与接口上的 IP 地址匹配的任何现有邻居条目，然后通过发送 IPv4 地址 (的 ARP 请求或 IPv6 地址的邻居请求来解析 MAC) 地址的物理地址。如果指定 `SourceAddress` 参数，`ResolveIpNetEntry2` 函数将选择要发送请求的具有此源 IP 地址的接口。如果在此参数中传递 `NULL` (未指定 `SourceAddress` 参数，`ResolveIpNetEntry2` 函数将自动选择要发送请求的最佳接口)。

`Row` 参数指向的 [MIB_IPNET_ROW2](#) 结构中的 `Address` 成员必须初始化为有效的 IPv4 或 IPv6 地址和系列。此外，指向 `Row` 参数的 [MIB_IPNET_ROW2](#) 结构中至少有一个成员必须初始化到接口：`InterfaceLuid` 或 `InterfaceIndex`。

字段按上面列出的顺序使用。因此，如果指定了 `InterfaceLuid`，则此成员用于确定要添加单播 IP 地址的接口。如果未为 `InterfaceLuid` 成员设置值 (此成员的值设置为零)，则接下来使用 `InterfaceIndex` 成员来确定接口。

如果 `Row` 参数指向的 [MIB_IPNET_ROW2](#) 的 `Address` 成员中传递的 IP 地址是接口上现有邻居 IP 地址的副本，则 `ResolveIpNetEntry2` 函数将在解析 IP 地址之前刷新现有条目。

当调用成功时，`ResolveIpNetEntry2` 在输出中检索邻居 IP 地址的其他属性，并填写 `Row` 参数指向的 [MIB_IPNET_ROW2](#) 结构。`Row` 参数指向的 [MIB_IPNET_ROW2](#) 结构中的 `PhysicalAddress` 和 `PhysicalAddressLength` 成员将初始化为有效的物理地址。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]

目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpNetEntry2](#)

[DeleteIpNetEntry2](#)

[FlushIpNetTable2](#)

[GetIpNetEntry2](#)

[GetIpNetTable2](#)

[MIB_IPNET_ROW2](#)

[MIB_IPNET_TABLE2](#)

[SOCKADDR_INET](#)

[SetIpNetEntry2](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

ResolveNeighbor 函数 (iphlpapi.h)

项目2023/08/24

[ResolveNeighbor 不再可用于 Windows Vista。 请改用 [ResolveIpNetEntry2](#)。]

ResolveNeighbor 函数解析本地计算机上邻居 IP 地址条目的物理地址。

语法

C++

```
IPHLPI_DLL_LINKAGE ULONG ResolveNeighbor(
    [in]      SOCKADDR *NetworkAddress,
    [out]     PVOID     PhysicalAddress,
    [in, out] PULONG   PhysicalAddressLength
);
```

parameters

[in] NetworkAddress

指向 [SOCKADDR](#) 结构的指针，该结构包含相邻 IP 地址条目和地址系列。

[out] PhysicalAddress

指向字节数组缓冲区的指针，如果函数成功，该缓冲区将接收与 *NetworkAddress* 参数指定的 IP 地址相对应的物理地址。字节数组的长度在 *PhysicalAddressLength* 参数中传递。

[in, out] PhysicalAddressLength

输入时，此参数指定 在 *PhysicalAddress* 参数中传递以接收物理地址的缓冲区的最大长度（以字节为单位）。如果函数成功，此参数将接收 *PhysicalAddress* 参数指向的缓冲区中返回的物理地址的长度。如果返回 [ERROR_BUFFER_OVERFLOW](#)，则此参数包含保存物理地址所需的字节数。

返回值

ResolveNeighbor 函数始终失败并返回以下错误代码。

返回代码	说明
ERROR_NOT_SUPPORTED	不支持该请求。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[ResolveIpNetEntry2](#)

[SOCKADDR](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

restoreMediaSense 函数 (iphlpapi.h)

项目2023/08/24

RestoreMediaSense 函数还原以前调用 DisableMediaSense 函数的本地计算机上的 TCP/IP 堆栈的媒体感知功能。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD RestoreMediaSense(
    OVERLAPPED *pOverlapped,
    [optional] LPDWORD lpdwEnableCount
);
```

parameters

pOverlapped

指向 OVERLAPPED 结构的指针。除 hEvent 成员外，此结构的所有成员都必须设置为零。 hEvent 成员应包含有效事件对象的句柄。使用 CreateEvent 函数创建此事件对象。

[optional] lpdwEnableCount

指向 DWORD 变量的可选指针，如果 RestoreMediaSense 函数成功，该变量接收剩余的引用数。变量也由 EnableRouter 和 UnenableRouter 函数使用。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 pOverlapped 参数是错误的指针，则返回此错误。如果在调用 RestoreMediaSense 函数之前未调用 DisableMediaSense 函数，也会返回此错误。
ERROR_IO_PENDING	操作正在进行中。对 RestoreMediaSense 的成功异步调用可能会返回此值。

ERROR_OPEN_FAILED	驱动程序的内部句柄无效。
ERROR_NOT_SUPPORTED	不支持该请求。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

如果 *pOverlapped* 参数为 NULL，则 **RestoreMediaSense** 函数将同步执行。

如果 *pOverlapped* 参数不为 NULL，则使用 *pOverlapped* 参数指向的 **OVERLAPPED** 结构异步执行 **RestoreMediaSense** 函数。

在稍后调用 **RestoreMediaSense** 函数以还原媒体感知功能之前，**DisableMediaSense** 函数不会完成。在此之前，I/O 请求数据包 (IRP) 保持排队状态。或者，当名为 **DisableMediaSense** 的进程退出时，将取消 IRP，并调用取消例程以再次还原媒体感知功能。

若要同步调用 **RestoreMediaSense**，应用程序需要在 *pOverlapped* 参数中传递 NULL 指针。同步调用 **RestoreMediaSense** 时，当 I/O 请求数据包 (IRP) 以还原媒体感知时，函数将返回。

若要异步调用 **RestoreMediaSense**，应用程序需要分配 **OVERLAPPED** 结构。除 *hEvent* 成员外，此结构的所有成员都必须设置为零。*hEvent* 成员需要有效事件对象的句柄。使用 [CreateEvent](#) 函数创建此事件。异步调用 **RestoreMediaSense** 时，可以返回 **ERROR_IO_PENDING**。当媒体感知功能恢复时，IRP 完成。当不再需要事件对象时，使用 [CloseHandle](#) 函数关闭该事件对象的句柄。进程终止时，系统会自动关闭句柄。事件对象在关闭其最后一个句柄时被销毁。

如果在调用 **RestoreMediaSense** 之前未调用 **DisableMediaSense**，则 **RestoreMediaSense** 返回**ERROR_INVALID_PARAMETER**。

在 Windows Server 2003 和 Windows XP 上，TCP/IP 堆栈实现删除接口上所有 IP 地址的策略，以响应与基础网络接口的媒体感知断开连接事件。如果本地计算机连接到的网络交换机或集线器已关闭电源，或者网络电缆断开连接，则网络接口将传送断开连接事件。与网络接口关联的 IP 配置信息丢失。因此，TCP/IP 堆栈实现隐藏断开连接的接口的策略，以便这些接口及其关联的 IP 地址不会显示在通过 IP 帮助程序检索的配置信息中。此策略可防止某些应用程序轻松检测网络接口是否只是断开连接，而不是从系统中删除。

如果本地客户端计算机使用 DHCP 请求来获取 IP 配置信息，则此行为通常不会影响本地客户端计算机。但这会对服务器计算机产生严重影响，尤其是用作群集一部分的计算机。**DisableMediaSense** 函数可用于暂时禁用这些情况下的媒体感知功能。稍后会调用 **RestoreMediaSense** 函数来还原媒体感知功能。

以下注册表设置与 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数相关：

系统\CurrentControlSet\服务\Tcpip\参数\DisableDHCPMediaSense

如果计算机首次启动时此注册表项存在，则 Windows 中会设置一个内部标志。也可以通过调用 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 来设置和重置相同的内部标志。但是，使用注册表设置时，需要重新启动计算机才能发生更改。

Windows Vista 及更高版本上的 TCP/IP 堆栈已更改为在发生断开连接事件时不隐藏断开连接的接口。因此，在 Windows Vista 及更高版本中，[DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数不执行任何操作，并且始终返回NO_ERROR。

示例

以下示例演示如何同步调用 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数。此示例仅适用于 Windows Server 2003 和 Windows XP，其中 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数执行了一些有用的操作。

该示例首先创建一个单独的线程，该线程同步调用 [DisableMediaSense](#) 函数，main线程休眠 60 秒以允许用户断开网络电缆的连接，检索 IP 地址表并打印表中 IP 地址条目的某些成员，同步调用 [RestoreMediaSense](#) 函数，再次检索 IP 地址表，和打印表中 IP 地址条目的一些成员。禁用媒体感知功能的影响可以从 IP 地址表条目的差异中看出。

有关演示如何异步调用 [DisableMediaSense](#) 和 [RestoreMediaSense](#) 函数的示例，请参阅 [DisableMediaSense](#) 函数参考。

C++

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

// The thread proc to call DisableMediaSense
DWORD WINAPI ThreadProc(LPVOID lpParam)
{
    if ((*((DWORD *) lpParam)) {
        DWORD dwRetVal;
        dwRetVal = DisableMediaSense(NULL, NULL);
        if (dwRetVal && dwRetVal != ERROR_IO_PENDING) {

```

```
        printf("DisableMediaSense failed with error %d\n", dwRetVal);
        return 0;
    } else {
        Sleep(1000);
        printf(" === DisableMediaSense Returned now. ===\n\n");
    }
}
return 0;
}

int __cdecl main()
{

    int i;

    /* Variables used by GetIpAddrTable */
    PMIB_IPADDRTABLE pIPAddrTable;
    DWORD dwSize = 0;
    DWORD dwRetVal = 0;
    IN_ADDR IPAddr;

    /* Variables used to return error message */
    LPVOID lpMsgBuf;

    /* Variable to use with RestoreMediaSense */
    DWORD dwEnableCount = 0;

    // Variables used to create a separate thread to call
    // the DisableMediaSense function
    DWORD ThreadID;
    DWORD IsDisable = TRUE;
    HANDLE Disable_THandle;

    // Create the thread to call Disable MediaSense synchronously
    Disable_THandle =
        CreateThread(NULL, 0, ThreadProc, (LPVOID) &IsDisable, 0,
&ThreadID);
    if (!Disable_THandle) {
        printf("CreateTread Failed:%d", GetLastError());
        exit(1);
    }

    printf(" === DisableMediaSense called on separate thread ===\n\n");
// Sleep for 60 seconds so we can disconnect a cable
    Sleep(60000);

    // Before calling AddIPAddress we use GetIpAddrTable to get
    // an adapter to which we can add the IP.
    pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(sizeof(MIB_IPADDRTABLE));

    if (pIPAddrTable) {
        // Make an initial call to GetIpAddrTable to get the
        // necessary size into the dwSize variable
        if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==
            ERROR_INSUFFICIENT_BUFFER) {


```

```

        FREE(pIPAddrTable);
        pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(dwSize);

    }

    if (pIPAddrTable == NULL) {
        printf("Memory allocation failed for GetIpAddrTable\n");
        exit(1);
    }

}

// Make a second call to GetIpAddrTable to get the
// actual data we want
if ((dwRetVal = GetIpAddrTable(pIPAddrTable, &dwSize, 0)) != NO_ERROR) {
    printf("GetIpAddrTable failed with error %d\n", dwRetVal);
    if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                      FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
                      NULL,
                      dwRetVal,
                      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default
language
                      (LPTSTR) & lpMsgBuf, 0, NULL)) {
        printf("\tError: %s", lpMsgBuf);
        LocalFree(lpMsgBuf);
    }
    exit(1);
}

printf("\tNum Entries: %ld\n", pIPAddrTable->dwNumEntries);
for (i = 0; i < (int) pIPAddrTable->dwNumEntries; i++) {
    printf("\n\tInterface Index[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwIndex);
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwAddr;
    printf("\tIP Address[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwMask;
    printf("\tSubnet Mask[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwBcastAddr;
    printf("\tBroadCast[%d]:\t%s (%ld)\n", i, inet_ntoa(IPAddr),
           pIPAddrTable->table[i].dwBcastAddr);
    printf("\tReassembly size[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwReasmSize);
    printf("\tType and State[%d]:", i);
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_PRIMARY)
        printf("\tPrimary IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DYNAMIC)
        printf("\tDynamic IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DISCONNECTED)
        printf("\tAddress is on disconnected interface");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DELETED)
        printf("\tAddress is being deleted");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_TRANSIENT)
        printf("\tTransient address");
    printf("\n");

}

// Call RestoreMediaSense synchronously to enable mediasense

```

```

dwRetVal = RestoreMediaSense(NULL, &dwEnableCount);
if (dwRetVal && dwRetVal != ERROR_IO_PENDING) {
    printf("RestoreMediaSense failed with error %d\n", dwRetVal);
    exit(1);
} else {
    printf(" === RestoreMediaSense called ===\n");
    printf("  EnableCount returned was %ld\n\n", dwEnableCount);
}

if (pIPAddrTable) {
    // Make an initial call to GetIpAddrTable to get the
    // necessary size into the dwSize variable
    if (GetIpAddrTable(pIPAddrTable, &dwSize, 0) ==
        ERROR_INSUFFICIENT_BUFFER) {
        FREE(pIPAddrTable);
        pIPAddrTable = (MIB_IPADDRTABLE *) MALLOC(dwSize);

    }
    if (pIPAddrTable == NULL) {
        printf("Memory allocation failed for GetIpAddrTable\n");
        exit(1);
    }
}
// Make a second call to GetIpAddrTable to get the
// actual data we want
if ((dwRetVal = GetIpAddrTable(pIPAddrTable, &dwSize, 0)) != NO_ERROR) {
    printf("GetIpAddrTable failed with error %d\n", dwRetVal);
    if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                      FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
                      NULL,
                      dwRetVal,
                      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default
language
                      (LPTSTR) & lpMsgBuf, 0, NULL)) {
        printf("\tError: %s", lpMsgBuf);
        LocalFree(lpMsgBuf);
    }
    exit(1);
}

printf("\tNum Entries: %ld\n", pIPAddrTable->dwNumEntries);
for (i = 0; i < (int) pIPAddrTable->dwNumEntries; i++) {
    printf("\n\tInterface Index[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwIndex);
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwAddr;
    printf("\tIP Address[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwMask;
    printf("\tSubnet Mask[%d]:\t%s\n", i, inet_ntoa(IPAddr));
    IPAddr.S_un.S_addr = (u_long) pIPAddrTable->table[i].dwBCastAddr;
    printf("\tBroadCast[%d]:\t%s (%ld)\n", i, inet_ntoa(IPAddr),
           pIPAddrTable->table[i].dwBCastAddr);
    printf("\tReassembly size[%d]:\t%ld\n", i,
           pIPAddrTable->table[i].dwReasmSize);
    printf("\tType and State[%d]:", i);
}

```

```

    if (pIPAddrTable->table[i].wType & MIB_IPADDR_PRIMARY)
        printf("\tPrimary IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DYNAMIC)
        printf("\tDynamic IP Address");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DISCONNECTED)
        printf("\tAddress is on disconnected interface");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_DELETED)
        printf("\tAddress is being deleted");
    if (pIPAddrTable->table[i].wType & MIB_IPADDR_TRANSIENT)
        printf("\tTransient address");
    printf("\n");
}

if (pIPAddrTable) {
    FREE(pIPAddrTable);
    pIPAddrTable = NULL;
}

exit(0);
}

```

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CloseHandle](#)

[CreateEvent](#)

[DisableMediaSense](#)

[EnableRouter](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[OVERLAPPED](#)

[UnenableRouter](#)

反馈

此页面是否有帮助?

[!\[\]\(352327531c09d1dba438eb09585e614d_img.jpg\) 是](#)

[!\[\]\(9442b48b3ef0c4fabfc24bc76fc57218_img.jpg\) 否](#)

[在 Microsoft Q&A 获得帮助](#)

RtlEthernetAddressToStringA 函数 (ip2string.h)

项目2024/01/27

RtlEthernetAddressToString 函数将二进制以太网地址转换为以太网 MAC 地址的字符串表示形式。

语法

C++

```
NTSYSAPI PSTR RtlEthernetAddressToStringA(
    [in] const DL_EUI48 *Addr,
    [out] PSTR           S
);
```

参数

[in] Addr

二进制格式的以太网地址。 以太网地址按网络顺序 (字节) 从左到右排序。

[out] S

指向用于存储以太网地址的 以 NULL 结尾的字符串表示形式的缓冲区的指针。 此缓冲区的大小应足以容纳至少 18 个字符。

返回值

指向插入到以太网 MAC 地址字符串表示形式的末尾的 NULL 字符的指针。 调用方可以使 用它轻松地将更多信息追加到字符串。

注解

RtlEthernetAddressToString 函数用于将二进制以太网地址转换为以太网 EUI-48 数据链路层地址格式的字符串表示形式，(通常也称为 MAC 地址)。 字符串表示以非 DIX 标准“-”表示法表示的数字以太网地址。

S 参数中返回的字符串以非 DIX 标准“-”表示法的以太网 MAC 地址字符串的形式表示。以太网 MAC 地址的基本字符串表示形式由 F4-CE-46-2D-90-8C (短划线分隔的 6 对十六进制数字组成，例如）。

RtlEthernetAddressToString 是一个便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行字符串到以太网地址的转换。

定义 **UNICODE** 或 **_UNICODE** 时，**RtlEthernetAddressToString** 将定义为 **RtlEthernetAddressToStringW**，即此函数的 Unicode 版本。字符串参数 *S* 和函数返回值定义为 **PWSTR** 数据类型。

如果未同时定义 **UNICODE** 和 **_UNICODE**，则 **RtlEthernetAddressToString** 将定义为 **RtlEthernetAddressToStringA**，即此函数的 ANSI 版本。字符串参数 *S* 和函数返回值定义为 **PSTR** 数据类型。

DL_EUI48 数据类型在 *Mstcpip.h* 头文件中定义。

① 备注

ip2string.h 标头将 **RtlEthernetAddressToString** 定义为别名，该别名根据 **UNICODE** 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。
有关详细信息，请参阅 [函数原型的约定](#)。

要求

[+] 展开表

最低受支持的客户端	Windows 7 [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 R2 [桌面应用 UWP 应用]
目标平台	Windows
标头	<i>ip2string.h</i> (包括 <i>Mstcpip.h</i> 、 <i>Ip2string.h</i>)
Library	<i>ntdll.lib</i>
DLL	<i>ntdll.dll</i>

另请参阅

反馈

此页面是否有帮助?

 是

 否

RtlEthernetStringToAddressA 函数 (ip2string.h)

项目2023/08/24

RtlEthernetStringToAddress 函数将以太网 MAC 地址的字符串表示形式转换为以太网地址的二进制格式。

语法

C++

```
NTSYSAPI NTSTATUS RtlEthernetStringToAddressA(
    [in]  PCSTR      S,
    [out] PCSTR     *Terminator,
    [out] DL_EUI48 *Addr
);
```

parameters

[in] S

指向缓冲区的指针，该缓冲区包含以 NULL 结尾的以太网 MAC 地址的字符串表示形式。

[out] Terminator

一个参数，该参数接收指向终止转换字符串的字符的指针。调用方可以使用它从字符串中提取更多信息。

[out] Addr

要存储以太网 MAC 地址的二进制表示形式的指针。

返回值

如果函数成功，则返回值 STATUS_SUCCESS。

如果函数失败，则返回值为以下错误代码之一。

返回代码

说明

STATUS_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>S</i> 参数指向的字符串不包含以太网 MAC 地址的正确字符串表示形式，则返回此错误。 此错误代码在 <i>Ntstatus.h</i> 头文件中定义。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`RtlEthernetStringToAddress` 函数用于将以太网 EUI-48 数据链路层地址的字符串表示形式转换为以太网地址（通常也称为 MAC 地址）为二进制格式的以太网地址。字符串表示以非 DIX 标准“-”表示法表示的数字以太网地址。返回的值是适合用作以太网地址的数字。所有以太网地址都按网络顺序返回，（字节从左到右）排序。

S 参数指向的字符串必须以非 DIX 标准“-”表示法表示以太网 MAC 地址字符串的形式表示。以太网 MAC 地址的基本字符串表示形式由 F4-CE-46-2D-90-8C（短划线分隔的 6 对十六进制数字组成，例如）。

成功后，*Terminator* 参数指向终止已转换的字符串的字符。这允许应用程序将包含以太网地址的字符串以及其他信息传递给 `RtlEthernetStringToAddress` 函数，然后分析其余信息。

`RtlEthernetStringToAddress` 是一个便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行字符串到以太网地址的转换。

定义 `UNICODE` 或 `_UNICODE` 时，`RtlEthernetStringToAddress` 将定义为 `RtlEthernetStringToAddressW`，即此函数的 Unicode 版本。*S* 和终止符参数定义为 `PCWSTR` 数据类型。

如果未定义 `UNICODE` 和 `_UNICODE`，则 `RtlEthernetStringToAddress` 将定义为 `RtlEthernetStringToAddressA`，即此函数的 ANSI 版本。*S* 和终止符参数定义为 `PCSTR` 数据类型。

`DL_EUI48` 数据类型在 *Mstcpip.h* 头文件中定义。

① 备注

`ip2string.h` 标头将 `RtlEthernetStringToAddress` 定义为别名，该别名根据 `UNICODE` 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

最低受支持的客户端	Windows 7 [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 R2 [桌面应用 UWP 应用]
目标平台	Windows
标头	ip2string.h (包括 Mstcpip.h、Ip2string.h)
DLL	Ntdll.dll

另请参阅

[RtlEthernetAddressToString](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

RtlIpv4AddressToStringA 函数 (ip2string.h)

项目2023/08/24

RtlIpv4AddressToString 函数将 IPv4 地址转换为 Internet 标准点十进制格式的字符串。

语法

C++

```
NTSYSAPI PSTR RtlIpv4AddressToStringA(
    [in] const in_addr *Addr,
    [out] PSTR      S
);
```

parameters

[in] Addr

按网络字节顺序排列的 IPv4 地址。

[out] S

指向缓冲区的指针，用于存储 IPv4 地址的以 NULL 结尾的字符串表示形式。此缓冲区应足够大，至少包含 16 个字符。

返回值

指向在 IPv4 地址的字符串表示形式末尾插入的 NULL 字符的指针。调用方可以使用它轻松将更多信息追加到字符串。

注解

RtlIpv4AddressToString 函数用于将 IPv4 地址转换为 Internet 点十进制格式的 IPv4 地址的字符串表示形式。

RtlIpv4AddressToString 是一种便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行 IP 地址到字符串的转换。

定义 UNICODE 或 _UNICODE 时，`RtlIpv4AddressToString` 将定义为 `RtlIpv4AddressToStringW`（此函数的 Unicode 版本）。字符串参数 `S` 和函数返回值定义为 PWSTR 数据类型。

如果未定义 UNICODE 和 _UNICODE，则 `RtlIpv4AddressToString` 将定义为 `RtlIpv4AddressToString`，即此函数的 ANSI 版本。字符串参数 `S` 和函数返回值定义为 PSTR 数据类型。

`IN_ADDR` 结构在 `Inaddr.h` 头文件中定义。

包含 `RtlIpv4AddressToString` 函数的导入库不包含在针对 Windows Vista 发布的 Microsoft Windows 软件开发工具包 (SDK) 中。`RtlIpv4AddressToString` 函数包含在 Windows 驱动程序工具包 (WDK) 中包含的 `Ntdll.lib` 导入库中。应用程序还可以使用 `GetModuleHandle` 和 `GetProcAddress` 函数从 `Ntdll.dll` 检索函数指针并调用此函数。

① 备注

`ip2string.h` 标头将 `RtlIpv4AddressToString` 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	<code>ip2string.h</code> (包括 <code>Mstcpip.h</code> 、 <code>Ip2string.h</code>)
DLL	<code>Ntdll.dll</code>

另请参阅

[GetModuleHandle](#)

[GetProcAddress](#)

[InetNtop](#)

[InetPton](#)

[LoadLibrary](#)

[RtlIpv4AddressToStringEx](#)

[RtlIpv4StringToAddress](#)

[RtlIpv4StringToAddressEx](#)

[RtlIpv6AddressToString](#)

[RtlIpv6AddressToStringEx](#)

[RtlIpv6StringToAddress](#)

[RtlIpv6StringToAddressEx](#)

[inet_addr](#)

[inet_ntoa](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

RtlIpv4AddressToStringExW 函数 (ip2string.h)

项目2024/01/27

RtlIpv4AddressToStringEx 函数将 IPv4 地址和端口号转换为 Internet 标准格式的字符串。

语法

C++

```
NTSYSAPI NTSTATUS RtlIpv4AddressToStringExW(
    [in]      const in_addr *Address,
    [in]      USHORT        Port,
    [out]     PWSTR         AddressString,
    [in, out] PULONG       AddressStringLength
);
```

参数

[in] Address

按网络字节顺序排列的 IPv4 地址。

[in] Port

网络字节顺序格式的端口号。此参数是可选的。

[out] AddressString

指向缓冲区的指针，用于接收 IPv4 地址和端口的以 NULL 结尾的字符串表示形式。此缓冲区应足够大，以容纳至少INET_ADDRSTRLEN个字符。INET_ADDRSTRLEN值在 Ws2ipdef.h 头文件中定义。

[in, out] AddressStringLength

输入时，AddressString 参数指向的缓冲区中适合的字符数，包括 NULL 终止符。在输出时，此参数包含实际写入到AddressString 参数指向的缓冲区的字符数。

返回值

如果函数成功，则返回值 STATUS_SUCCESS。

如果函数失败，则返回值为以下错误代码之一。

[] 展开表

返回代码	说明
STATUS_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>AddressString</i> 或 <i>AddressStringLength</i> 参数中传递 NULL 指针，则返回此错误。如果 <i>AddressString</i> 参数指向的缓冲区长度不足以接收 IPv4 地址和端口的字符串表示形式，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

RtlIpv4AddressToStringEx 函数用于将 IPv4 地址和端口号转换为 Internet 虚线十进制格式的 IPv4 地址的字符串表示形式，后跟冒号字符和端口的字符串表示形式。

RtlIpv4AddressToStringEx 是一种便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行 IP 地址到字符串的转换。

如果 *AddressString* 参数指向的缓冲区长度不足以接收 IPv4 地址和端口的字符串表示形式，则 **RtlIpv4AddressToStringEx** 将返回 ERROR_INVALID_PARAMETER 并将 *AddressStringLength* 参数设置为所需的缓冲区长度。

定义 UNICODE 或 _UNICODE 时，**RtlIpv4AddressToStringEx** 将定义为 **RtlIpv4AddressToStringExW**（此函数的 Unicode 版本）。*AddressString* 参数定义为 PWSTR 数据类型。

如果未定义 UNICODE 和 _UNICODE，则 **RtlIpv4AddressToStringEx** 将定义为 **RtlIpv4AddressToStringExA**（此函数的 ANSI 版本）。*AddressString* 参数定义为 PSTR 数据类型。

IN_ADDR 结构在 *Inaddr.h* 头文件中定义。

包含 **RtlIpv4AddressToStringEx** 函数的导入库不包含在针对 Windows Vista 发布的 Microsoft Windows 软件开发工具包 (SDK) 中。**RtlIpv4AddressToStringEx** 函数包含在 Windows 驱动程序工具包 (WDK) 中包含的 *Ntdll.lib* 导入库中。应用程序还可以使用 [GetModuleHandle](#) 和 [GetProcAddress](#) 函数从 *Ntdll.dll* 检索函数指针并调用此函数。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	ip2string.h (包括 Mstcpip.h)
Library	ntdll.lib
DLL	ntdll.dll

另请参阅

[GetModuleHandle](#)

[GetProcAddress](#)

[InetNtop](#)

[InetPton](#)

[LoadLibrary](#)

[RtlIpv4AddressToString](#)

[RtlIpv4StringToAddress](#)

[RtlIpv4StringToAddressEx](#)

[RtlIpv6AddressToString](#)

[RtlIpv6AddressToStringEx](#)

[RtlIpv6StringToAddress](#)

[RtlIpv6StringToAddressEx](#)

[inet_addr](#)

[inet_ntoa](#)

反馈

此页面是否有帮助?

是

否

RtlIpv6AddressToStringA 函数 (ip2string.h)

项目2024/01/27

RtlIpv6AddressToString 函数将 IPv6 地址转换为 Internet 标准格式的字符串。

语法

C++

```
NTSYSAPI PSTR RtlIpv6AddressToStringA(
    [in] const in6_addr *Addr,
    [out] PSTR           S
);
```

参数

[in] Addr

按网络字节顺序排列的 IPv6 地址。

[out] S

指向存储 IPv6 地址的以 NULL 结尾的字符串表示形式的缓冲区的指针。此缓冲区的大小应足以容纳至少 46 个字符。

返回值

指向在 IPv6 地址的字符串表示形式末尾插入的 NULL 字符的指针。调用方可以使用它轻松地将更多信息追加到字符串。

注解

RtlIpv6AddressToString 函数用于将 IPv6 地址转换为 Internet 标准格式的 IPv6 地址的字符串表示形式。

基本字符串表示形式由 8 个用冒号分隔的十六进制数字组成。由连续零数字的字符串替换为双冒号。IPv6 地址的字符串表示形式中只能有一个双冒号。如果地址是 IPv4 兼容

地址、IPv4 映射的 IPv6 地址或 ISATAP 地址，则最后 32 位以 IPv4 样式的点位八进制表示法表示。有关详细信息，请参阅 IETF 发布的 [RFC 5942 的第 5 部分](#)。

RtIpv6AddressToString 是一个便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行 IP 地址到字符串的转换。

定义 **UNICODE** 或 **_UNICODE** 时，**RtIpv6AddressToString** 将定义为 **RtIpv6AddressToStringW**，即此函数的 Unicode 版本。字符串参数 *S* 和函数返回值定义为 **PWSTR** 数据类型。

如果未定义 **UNICODE** 和 **_UNICODE**，则 **RtIpv6AddressToString** 将定义为 **RtIpv6AddressToStringA**，即此函数的 ANSI 版本。字符串参数 *S* 和函数返回值定义为 **PSTR** 数据类型。

IN6_ADDR 结构在 *In6addr.h* 头文件中定义。

包含 **RtIpv6AddressToString** 函数的导入库不包含在针对 Windows Vista 发布的 Microsoft Windows 软件开发工具包 (SDK) 中。**RtIpv6AddressToString** 函数包含在 Windows 驱动程序工具包 (WDK) 中包含的 *Ntdll.lib* 导入库中。应用程序还可以使用 **GetModuleHandle** 和 **GetProcAddress** 函数从 *Ntdll.dll* 检索函数指针并调用此函数。

① 备注

ip2string.h 标头将 **RtIpv6AddressToString** 定义为别名，该别名根据 **UNICODE** 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

[] 展开表

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	<i>ip2string.h</i> (包括 <i>Mstcpip.h</i> 、 <i>Ip2String.h</i>)
Library	<i>ntdll.lib</i>

DLL	ntdll.dll
-----	-----------

另请参阅

[GetModuleHandle](#)

[GetProcAddress](#)

[InetNtop](#)

[InetPton](#)

[LoadLibrary](#)

[RtlIpv4AddressToString](#)

[RtlIpv4AddressToStringEx](#)

[RtlIpv4StringToAddress](#)

[RtlIpv4StringToAddressEx](#)

[RtlIpv6AddressToStringEx](#)

[RtlIpv6StringToAddress](#)

[RtlIpv6StringToAddressEx](#)

[inet_addr](#)

[inet_ntoa](#)

反馈

此页面是否有帮助?

是

否

RtlIpv6AddressToStringExW 函数 (ip2string.h)

项目2024/01/27

RtlIpv6AddressToStringEx 函数将 IPv6 地址、范围 ID 和端口号转换为字符串。

语法

C++

```
NTSYSAPI NTSTATUS RtlIpv6AddressToStringExW(
    [in]      const in6_addr *Address,
    [in]      ULONG        ScopeId,
    [in]      USHORT       Port,
    [out]     PWSTR        AddressString,
    [in, out] PULONG      AddressStringLength
);
```

参数

[in] Address

按网络字节顺序排列的 IPv6 地址。

[in] ScopeId

按网络字节顺序排列的 IPv6 地址的范围 ID。此参数是可选的。

[in] Port

网络字节顺序格式的端口号。此参数是可选的。

[out] AddressString

指向缓冲区的指针，用于接收 IP 地址、范围 ID 和端口的以 NULL 结尾的字符串表示形式。此缓冲区应足够大，以容纳至少INET6_ADDRSTRLEN个字符。INET6_ADDRSTRLEN 值在 *Ws2ipdef.h* 头文件中定义。

[in, out] AddressStringLength

输入时，*AddressString* 参数指向的缓冲区中适合的字符数，包括 NULL 终止符。在输出时，此参数包含实际写入到 *AddressString* 参数指向的缓冲区的字符数。

返回值

如果函数成功，则返回值 STATUS_SUCCESS。

如果函数失败，则返回值为以下错误代码之一。

[+] 展开表

返回代码	说明
STATUS_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>AddressString</i> 或 <i>AddressStringLength</i> 参数中传递 NULL 指针，则返回此错误。如果 <i>AddressString</i> 参数指向的缓冲区长度不足以接收 IPv6 地址、范围 ID 和端口号的字符串表示形式，则也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

RtIpv6AddressToStringEx 函数用于将 IPv6 地址、范围 ID 和端口号转换为 Internet 格式的 IPv6 地址的字符串表示形式，后跟范围 ID 的字符串表示形式，后跟端口的字符串表示形式。范围 ID 和端口号是可选参数。

返回的 IPv6 地址的基本字符串表示形式由 8 个用冒号分隔的十六进制数组成。连续零个十六进制数字的字符串将替换为双冒号。IPv6 地址的字符串表示形式中只能有一个双冒号。如果地址是 IPv4 兼容地址、IPv4 映射的 IPv6 地址或 ISATAP 地址，则最后 32 位以 IPv4 样式的点八进制表示法表示。有关详细信息，请参阅 IETF 发布的 [RFC 5942 第 5 部分](#)。

如果提供了范围 ID，则范围 ID 的字符串表示形式与 IPv6 地址的字符串表示形式 ('%') 分隔。如果提供了端口号，则 IPv6 地址和范围 ID 的字符串表示形式用方括号括起来，(前导的 "[" 字符，后跟 IPv6 地址，后跟一个 "% " 字符，后跟一个带有尾随 "]" 字符的范围 ID)。端口号表示为一个冒号，后跟右方括号字符，后跟端口号的字符串表示形式 (以十进制为单位)。

RtIpv6AddressToStringEx 是一个方便函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行 IP 地址到字符串的转换。

如果 *AddressString* 参数指向的缓冲区长度不足以接收 IP 地址、范围 ID 和端口号的字符串表示形式，则 **RtIpv6AddressToStringEx** 将返回 ERROR_INVALID_PARAMETER 并将 *AddressStringLength* 参数设置为所需的缓冲区长度。

定义 UNICODE 或 _UNICODE 时，`RtlIpv6AddressToStringEx` 将定义为 `RtlIpv6AddressToStringExW`，即此函数的 Unicode 版本。`AddressString` 参数定义为 PWSTR 数据类型。

如果未定义 UNICODE 和 _UNICODE，则 `RtlIpv6AddressToStringEx` 定义为 `RtlIpv6AddressToStringExA`（此函数的 ANSI 版本）。`AddressString` 参数定义为 PSTR 数据类型。

`IN6_ADDR` 结构在 `In6addr.h` 头文件中定义。

包含 `RtlIpv6AddressToStringEx` 函数的导入库不包含在针对 Windows Vista 发布的 Microsoft Windows 软件开发工具包 (SDK) 中。`RtlIpv6AddressToStringEx` 函数包含在 Windows 驱动程序工具包 (WDK) 中包含的 `Ntdll.lib` 导入库中。应用程序还可以使用 `GetModuleHandle` 和 `GetProcAddress` 函数从 `Ntdll.dll` 检索函数指针并调用此函数。

要求

[] 展开表

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	<code>ip2string.h</code> (包括 <code>Mstcpip.h</code>)
Library	<code>ntdll.lib</code>
DLL	<code>ntdll.dll</code>

另请参阅

[GetModuleHandle](#)

[GetProcAddress](#)

[InetNtop](#)

[InetPton](#)

[LoadLibrary](#)

[RtlIpv4AddressToString](#)

[RtlIpv4AddressToStringEx](#)

[RtlIpv4StringToAddress](#)

[RtlIpv4StringToAddressEx](#)

[RtlIpv6AddressToString](#)

[RtlIpv6StringToAddress](#)

[RtlIpv6StringToAddressEx](#)

[inet_addr](#)

[inet_ntoa](#)

反馈

此页面是否有帮助?

 是

 否

RtlIpv4StringToAddressA 函数 (ip2string.h)

项目2023/08/24

RtlIpv4StringToAddress 函数将 IPv4 地址的字符串表示形式转换为二进制 IPv4 地址。

语法

C++

```
NTSYSAPI NTSTATUS RtlIpv4StringToAddressA(
    [in] PCSTR S,
    [in] BOOLEAN Strict,
    [out] PCSTR *Terminator,
    [out] in_addr *Addr
);
```

parameters

[in] S

指向包含 IPv4 地址以 NULL 结尾的字符串表示形式的缓冲区的指针。

[in] Strict

一个值，该值指示字符串是否必须是以严格的四部分点十进制表示法表示的 IPv4 地址。如果此参数为 TRUE，则字符串必须是带四个部分的点十进制。如果此参数为 FALSE，则允许使用十进制、八进制或十六进制表示法的四种可能形式中的任何一种。有关详细信息，请参见“备注”部分。

[out] Terminator

一个参数，用于接收指向终止转换字符串的字符的指针。调用方可以使用它从字符串中提取更多信息。

[out] Addr

要存储 IPv4 地址的二进制表示形式的指针。

返回值

如果函数成功，则返回值 STATUS_SUCCESS。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
STATUS_INVALID_PARAMETER	向该函数传递了无效参数。如果 Strict 参数设置为 TRUE，但 S 参数指向的字符串不包含 IPv4 地址的四部分点十进制字符串表示形式，则返回此错误。如果 S 参数指向的字符串不包含 IPv4 地址的正确字符串表示形式，则也会返回此错误。 此错误代码在 Ntstatus.h 头文件中定义。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`RtlIpv4StringToAddress` 函数用于将 IPv4 地址的字符串表示形式转换为按网络顺序返回的 IPv4 地址，(字节从左到右) 排序。

`RtlIpv4StringToAddress` 是一种便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行字符串到 IP 地址的转换。

如果 `Strict` 参数设置为 TRUE，则 `S` 参数指向的字符串必须采用严格的点十进制表示法。这种严格的格式要求指定四个部分。每个部分都解释为数据的十进制字节，并从左到右分配给 IPv4 地址的四个字节。

如果将 `Strict` 参数设置为 FALSE，则 `S` 参数指向的字符串可能采用多种可能的格式之一。当 `S` 参数指向的缓冲区包含一个由三部分构成的地址字符串时，最后一部分将被解释为 16 位数量，并放置在网络地址的右边最多两个字节中。这使得由三部分构成的地址格式便于将 B 类网络地址指定为“128.net.host”。当 `S` 参数指向的缓冲区包含一个由两部分构成的地址字符串时，最后一部分将被解释为 24 位数量，并放置在网络地址的右三个字节中。这使得两部分地址格式便于将类 A 网络地址指定为“net.host”。当 `S` 参数指向的缓冲区仅包含一部分地址字符串时，该值将直接存储在网络地址中，而无需重新排列任何字节。

成功后，`终止符` 参数指向终止转换的字符串的字符。这样，应用程序就可以将包含 IP 地址和其他信息的字符串传递给 `RtlIpv4StringToAddress` 函数，然后分析其余信息。

定义 UNICODE 或 _UNICODE 时，`RtlIpv4StringToAddress` 将定义为 `RtlIpv4StringToAddressW` (此函数的 Unicode 版本)。`S` 参数定义为 PCWSTR 数据类型，`终止符` 参数定义为 LPCWSTR 数据类型。

如果未定义 UNICODE 和 _UNICODE，则 `RtlIpv4StringToAddress` 将定义为 `RtlIpv4StringToAddressA`（此函数的 ANSI 版本）。`S` 和终止符参数定义为 PCSTR 数据类型。

`IN_ADDR` 结构在 `Inaddr.h` 头文件中定义。

包含 `RtlIpv4StringToAddress` 函数的导入库不包含在针对 Windows Vista 发布的 Microsoft Windows 软件开发工具包 (SDK) 中。`RtlIpv4StringToAddress` 函数包含在 Windows 驱动程序工具包 (WDK) 中包含的 `Ntdll.lib` 导入库中。应用程序还可以使用 `GetModuleHandle` 和 `GetProcAddress` 函数从 `Ntdll.dll` 检索函数指针并调用此函数。

① 备注

`ip2string.h` 标头将 `RtlIpv4StringToAddress` 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名的使用与非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	<code>ip2string.h</code> (包括 <code>Mstcpip.h</code> 、 <code>Ip2string.h</code>)
DLL	<code>Ntdll.dll</code>

另请参阅

[GetModuleHandle](#)

[GetProcAddress](#)

[InetNtop](#)

[InetPton](#)

[LoadLibrary](#)

[RtlIpv4AddressToString](#)

[RtlIpv4AddressToStringEx](#)

[RtlIpv4StringToAddressEx](#)

[RtlIpv6AddressToString](#)

[RtlIpv6AddressToStringEx](#)

[RtlIpv6StringToAddress](#)

[RtlIpv6StringToAddressEx](#)

[inet_addr](#)

[inet_ntoa](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

RtlIpv4StringToAddressExW 函数 (ip2string.h)

项目2023/08/24

RtlIpv4StringToAddressEx 函数将 IPv4 地址和端口号的字符串表示形式转换为二进制 IPv4 地址和端口。

语法

C++

```
NTSYSAPI NTSTATUS RtlIpv4StringToAddressExW(
    [in] PCWSTR AddressString,
    [in] BOOLEAN Strict,
    [out] in_addr *Address,
    [out] PUSHORT Port
);
```

parameters

[in] AddressString

指向缓冲区的指针，该缓冲区包含以 NULL 结尾的 IPv4 地址字符串表示形式，后跟端口号的可选冒号和字符串表示形式。

[in] Strict

一个值，该值指示字符串是否必须是以严格的四部分点十进制表示法表示的 IPv4 地址。如果此参数为 TRUE，则字符串必须带有四个部分的点-小数。如果此参数为 FALSE，则 IPv4 地址的字符串表示形式允许有四种形式中的任何一种，使用十进制、八进制或十六进制表示法。有关详细信息，请参见“备注”部分。

[out] Address

要存储 IPv4 地址的二进制表示形式的指针。IPv4 地址按网络字节顺序存储。

[out] Port

要存储端口号的二进制表示形式的指针。端口号按网络字节顺序返回。如果在 AddressString 参数指向的字符串中未指定端口，则 Port 参数设置为零。

返回值

如果函数成功，则返回值 STATUS_SUCCESS。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
STATUS_INVALID_PARAMETER	向该函数传递了无效参数。如果 Strict 参数设置为 TRUE，但 AddressString 参数指向的字符串不包含 IPv4 地址的四部分点十进制字符串表示形式，则返回此错误。如果 AddressString 参数指向的字符串不包含 IPv4 地址的正确字符串表示形式，也会返回此错误。 此错误代码在 Ntstatus.h 头文件中定义。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

RtlIpv4StringToAddressEx 函数用于将 IPv4 地址和端口号的字符串表示形式转换为二进制 IPv4 地址和端口号。IPv4 地址按网络顺序返回，(字节从左到右) 排序。端口号按网络顺序返回。

RtlIpv4StringToAddressEx 是一个便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行字符串到 IP 地址的转换。

如果 Strict 参数设置为 TRUE，则 AddressString 参数指向的字符串必须采用严格的点十进制表示法。此严格格式要求指定四个部分。每个部分解释为数据的十进制字节，并从左到右分配给 IPv4 地址的四个字节。

当 Strict 参数设置为 FALSE 时，AddressString 参数指向的字符串可能采用多种可能格式中的任何一种。当 AddressString 参数指向的缓冲区包含由三部分构成的地址字符串时，最后一部分将解释为 16 位数量，并放置在网络地址的右侧最多两个字节中。这使得三部分地址格式便于将 B 类网络地址指定为“128.net.host”。当 AddressString 参数指向的缓冲区包含由两部分构成的地址字符串时，最后一部分将被解释为 24 位数量并放置在网络地址的最多三个字节中。这使得两部分地址格式便于将类 A 网络地址指定为“net.host”。当 AddressString 参数指向的缓冲区仅包含一部分地址字符串时，该值直接存储在网络地址中，无需任何字节重新排列。

AddressString 参数指向的缓冲区可能包含 IPv4 地址字符串，后跟可选的冒号和端口号的字符串表示形式。如果 AddressString 参数指向的缓冲区中包含端口号字符串，则会在 Port 参数中返回端口号的二进制表示形式。如果 AddressString 参数指向的缓冲区不包含端口号，则 Port 参数中返回零。

定义 UNICODE 或 _UNICODE 时，`RtlIpv4StringToAddressEx` 将定义为 `RtlIpv4StringToAddressExW`，即此函数的 Unicode 版本。`AddressString` 参数定义为 PCWSTR 数据类型。

如果未同时定义 UNICODE 和 _UNICODE，则 `RtlIpv4StringToAddressEx` 将定义为 `RtlIpv4StringToAddressExA`，即此函数的 ANSI 版本。`AddressString` 参数定义为 PCSTR 数据类型。

`IN_ADDR` 结构在 `Inaddr.h` 头文件中定义。

包含 `RtlIpv4StringToAddressEx` 函数的导入库不包含在针对 Windows Vista 发布的 Microsoft Windows 软件开发工具包 (SDK) 中。`RtlIpv4StringToAddressEx` 函数包含在 Windows 驱动程序工具包 (WDK) 中包含的 `Ntdll.lib` 导入库中。应用程序还可以使用 `GetModuleHandle` 和 `GetProcAddress` 函数从 `Ntdll.dll` 检索函数指针并调用此函数。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	<code>ip2string.h</code> (包括 <code>Mstcpip.h</code>)
DLL	<code>Ntdll.dll</code>

另请参阅

[GetModuleHandle](#)

[GetProcAddress](#)

[IN_ADDR](#)

[InetNtop](#)

[InetPton](#)

[LoadLibrary](#)

[RtlIpv4AddressToString](#)

[RtlIpv4AddressToStringEx](#)

[RtlIpv4StringToAddress](#)

[RtlIpv6AddressToString](#)

[RtlIpv6AddressToStringEx](#)

[RtlIpv6StringToAddress](#)

[RtlIpv6StringToAddressEx](#)

[inet_addr](#)

[inet_ntoa](#)

反馈

此页面是否有帮助?

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

RtlIpv6StringToAddressA 函数 (ip2string.h)

项目2023/08/24

RtlIpv6StringToAddress 函数将 IPv6 地址的字符串表示形式转换为二进制 IPv6 地址。

语法

C++

```
NTSYSAPI NTSTATUS RtlIpv6StringToAddressA(
    [in]  PCSTR      S,
    [out] PCSTR     *Terminator,
    [out] in6_addr  *Addr
);
```

parameters

[in] S

指向缓冲区的指针，该缓冲区包含以 NULL 结尾的 IPv6 地址的字符串表示形式。

[out] Terminator

一个参数，该参数接收指向终止转换字符串的字符的指针。调用方可以使用它从字符串中提取更多信息。

[out] Addr

要存储 IPv6 地址的二进制表示形式的指针。

返回值

如果函数成功，则返回值 STATUS_SUCCESS。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
STATUS_INVALID_PARAMETER	向该函数传递了无效参数。如果 S 参数指向的字符串不包含 IPv6 地址的正确字符串表示形式，则返回此错误。

此错误代码在 Ntstatus.h 头文件中定义。

其他

使用 [FormatMessage](#) 获取返回错误的消息字符串。

注解

`RtlIpv6StringToAddress` 函数用于将 IPv6 地址的字符串表示形式转换为按网络顺序返回的 IPv6 地址，(字节从左到右) 排序。

`RtlIpv6StringToAddress` 是一个便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数，以执行字符串到 IP 地址的转换。

`S` 参数指向的字符串必须以 IPv6 地址字符串的形式表示。IPv6 地址的基本字符串表示形式由 8 个用冒号分隔的十六进制数字组成。连续零数字的字符串可以替换为双冒号。

IPv6 地址的字符串表示形式中只能有一个双冒号。如果地址是 IPv4 兼容地址、IPv4 映射的 IPv6 地址或 ISATAP 地址，则最后 32 位可以以 IPv4 样式的点位八位表示法表示。有关详细信息，请参阅 IETF 发布的 [RFC 5942 的第 5 部分](#)。

成功后，`Terminator` 参数指向终止已转换的字符串的字符。这允许应用程序将包含和 IP 地址以及其他信息的字符串传递给 `RtlIpv6StringToAddress` 函数，然后分析其余信息。

注意 某些格式错误的 IPv6 地址 (: : : : , 例如,) 以有效的 IPv6 地址开头。

`RtlIpv6StringToAddress` 函数将返回成功，并将 IPv6 地址的有效部分解析为双冒号 (: : : :)。终止符随后指向第三个冒号。若要验证整个传入字符串是否为有效的 IPv6 地址，需要确保终止符指向正确的字符。如果 `S` 参数仅包含 IPv6 地址，则终止符应指向字符串末尾的 NULL 字符。

定义 `UNICODE` 或 `_UNICODE` 时，`RtlIpv6StringToAddress` 将定义为 `RtlIpv6StringToAddressW`，即此函数的 Unicode 版本。`S` 和终止符参数定义为 `PCWSTR` 数据类型。

如果未同时定义 `UNICODE` 和 `_UNICODE`，则 `RtlIpv6StringToAddress` 将定义为 `RtlIpv6StringToAddressA`，即此函数的 ANSI 版本。`S` 和终止符参数定义为 `PCSTR` 数据类型。

`IN6_ADDR` 结构在 `In6addr.h` 头文件中定义。

包含 **RtlIpv6StringToAddress 函数的** 导入库不包含在针对 Windows Vista 发布的 Microsoft Windows 软件开发工具包 (SDK) 中。`RtlIpv6StringToAddress` 函数包含在 Windows 驱动程序工具包 (WDK) 中包含的 `Ntdll.lib` 导入库中。应用程序还可以使用 `GetModuleHandle` 和 `GetProcAddress` 函数从 `Ntdll.dll` 检索函数指针并调用此函数。

① 备注

ip2string.h 标头将 RtlIpv6StringToAddress 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。将非特定编码别名与非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。有关详细信息，请参阅 [函数原型的约定](#)。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	ip2string.h (包括 Mstcpip.h、Ip2string.h)
DLL	Ntdll.dll

另请参阅

[GetModuleHandle](#)

[GetProcAddress](#)

[IN6_ADDR](#)

[InetNtop](#)

[InetPton](#)

[LoadLibrary](#)

[RtlIpv4AddressToString](#)

[RtlIpv4AddressToStringEx](#)

[RtlIpv4StringToAddress](#)

[RtlIpv4StringToAddressEx](#)

[RtlIpv6AddressToString](#)

[RtlIpv6AddressToStringEx](#)

[RtlIpv6StringToAddressEx](#)

[inet_addr](#)

[inet_ntoa](#)

反馈

此页面是否有帮助？

[是](#)

[否](#)

[在 Microsoft Q&A 获得帮助](#)

RtlIpv6StringToAddressExW 函数 (ip2string.h)

项目2023/08/24

RtlIpv6StringToAddressEx 函数将 IPv6 地址、范围 ID 和端口号的字符串表示形式转换为二进制 IPv6 地址、范围 ID 和端口。

语法

C++

```
NTSYSAPI NTSTATUS RtlIpv6StringToAddressExW(
    [in] PCWSTR AddressString,
    [out] in6_addr *Address,
    [out] PULONG ScopeId,
    [out] PUSHORT Port
);
```

parameters

[in] AddressString

指向缓冲区的指针，该缓冲区包含以 NULL 结尾的 IPv6 地址、范围 ID 和端口号的字符串表示形式。

[out] Address

要存储 IPv6 地址的二进制表示形式的指针。

[out] ScopeId

指向存储 IPv6 地址的范围 ID 的指针。如果 *AddressString* 参数不包含范围 ID 的字符串表示形式，则此参数中返回零。

[out] Port

存储端口号的指针。端口号采用网络字节顺序格式。如果 *AddressString* 参数不包含端口号的字符串表示形式，则此参数中返回零。

返回值

如果函数成功，则返回值 STATUS_SUCCESS。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
STATUS_INVALID_PARAMETER	向该函数传递了无效参数。如果 <i>AddressString</i> 参数指向的字符串不包含 IPv6 地址的正确字符串表示形式，则返回此错误。 此错误代码在 Ntstatus.h 头文件中定义。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

RtlIpv6StringToAddressEx 函数用于将 IPv6 地址、范围 ID 和端口号的字符串表示形式转换为二进制 IPv6 地址、范围 ID 和端口号。IPv6 地址按网络顺序返回，(字节从左到右排序)。端口号和范围 ID 按网络顺序返回。

RtlIpv6StringToAddressEx 是一个便捷函数，它不需要加载 Windows 套接字 DLL 即可访问 Windows 套接字中提供的函数以执行字符串到 IP 地址的转换。

AddressString 参数指向的字符串必须以 IPv6 地址字符串的形式表示，后跟可选的百分比字符和范围 ID 字符串。IPv6 地址和范围 ID 字符串必须括在方括号中。IPv6 地址和范围 ID 字符串后面的右方括号可以后跟可选的冒号和端口号的字符串表示形式。IPv6 地址的基本字符串表示形式由 8 个用冒号分隔的十六进制数字组成。连续零数字的字符串可以替换为双冒号。IPv6 地址的字符串表示形式中只能有一个双冒号。如果地址是 IPv4 兼容地址、IPv4 映射的 IPv6 地址或 ISATAP 地址，则最后 32 位可以以 IPv4 样式的点位八位表示法表示。有关详细信息，请参阅 IETF 发布的 [RFC 5942 的第 5 部分](#)。

定义 UNICODE 或 _UNICODE 时，**RtlIpv6StringToAddressEx** 将定义为 **RtlIpv6StringToAddressExW**，即此函数的 Unicode 版本。*AddressString* 参数定义为 PCWSTR 数据类型。

如果未同时定义 UNICODE 和 _UNICODE，则 **RtlIpv6StringToAddressEx** 将定义为 **RtlIpv6StringToAddressExA**，即此函数的 ANSI 版本。*AddressString* 参数定义为 PCSTR 数据类型。

[IN6_ADDR](#) 结构在 In6addr.h 头文件中定义。

包含 **RtlIpv6StringToAddressEx** 函数的导入库不包含在针对 Windows Vista 发布的 Microsoft Windows 软件开发工具包 (SDK) 中。**RtlIpv6StringToAddressEx** 函数包含在 Windows 驱动程序工具包 (WDK) 中包含的 *Ntdll.lib* 导入库中。应用程序还可以使用 [GetModuleHandle](#) 和 [GetProcAddress](#) 函数从 *Ntdll.dll* 检索函数指针并调用此函数。

要求

最低受支持的客户端	Windows Vista [桌面应用 UWP 应用]
最低受支持的服务器	Windows Server 2008 [桌面应用 UWP 应用]
目标平台	Windows
标头	ip2string.h (包括 Mstcpip.h)
DLL	Ntdll.dll

另请参阅

[GetModuleHandle](#)

[GetProcAddress](#)

[IN6_ADDR](#)

[InetNtop](#)

[InetPton](#)

[LoadLibrary](#)

[RtlIpv4AddressToString](#)

[RtlIpv4AddressToStringEx](#)

[RtlIpv4StringToAddress](#)

[RtlIpv4StringToAddressEx](#)

[RtlIpv6AddressToString](#)

[RtlIpv6AddressToStringEx](#)

[RtlIpv6StringToAddress](#)

[inet_addr](#)

[inet_ntoa](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

SendARP 函数 (iphlpapi.h)

项目2023/08/24

SendARP 函数 (ARP) 请求发送地址解析协议，以获取与指定目标 IPv4 地址对应的物理地址。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD SendARP(
    [in]      IPAddr DestIP,
    [in]      IPAddr SrcIP,
    [out]     PVOID pMacAddr,
    [in, out] PULONG PhyAddrLen
);
```

parameters

[in] DestIP

目标 IPv4 地址，采用 [IPAddr](#) 结构的形式。 ARP 请求尝试获取与此 IPv4 地址对应的物理地址。

[in] SrcIP

发送方的源 IPv4 地址，采用 [IPAddr](#) 结构的形式。此参数是可选的，用于选择接口以针对 ARP 条目发送请求。调用方可以指定与此参数的 INADDR_ANY IPv4 地址对应的零。

[out] pMacAddr

指向 ULONG 变量数组的指针。此数组必须至少有两个 ULONG 元素才能保存以太网或令牌环物理地址。此数组的前六个字节接收与 *DestIP* 参数指定的 IPv4 地址对应的物理地址。

[in, out] PhyAddrLen

输入时，一个指向 ULONG 值的指针，该值指定最大缓冲区大小（以字节为单位），应用程序已留出一部分来接收物理地址或 MAC 地址。对于以太网或令牌环物理地址，缓冲区大小应至少为 6 个字节

用于接收物理地址的缓冲区由 *pMacAddr* 参数指向。

成功输出时，此参数指向一个值，该值指定写入 *pMacAddr* 指向的缓冲区的字节数。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_BAD_NET_NAME	找不到网络名称。当未收到对 SendARP 请求的 ARP 答复时，将在 Windows Vista 和更高版本上返回此错误。如果由于目标 IPv4 地址不在同一子网中或目标计算机未运行而无法访问，则会发生此错误。
ERROR_BUFFER_OVERFLOW	文件名太长。如果 <i>PhyAddrLen</i> 参数指向的 ULONG 值小于 6（存储完整物理地址所需的大小），则会在 Windows Vista 上返回此错误。
ERROR_GEN_FAILURE	附加到系统的设备无法正常工作。如果未收到对 SendARP 请求的 ARP 答复，则会在 Windows Server 2003 及更早版本上返回此错误。如果由于目标 IPv4 地址不在同一子网中或目标计算机无法运行而无法访问，则会发生此错误。
ERROR_INVALID_PARAMETER	其中一个参数无效。如果 <i>pMacAddr</i> 或 <i>PhyAddrLen</i> 参数为 NULL 指针，则会在 Windows Server 2003 及更早版本上返回此错误。
ERROR_INVALID_USER_BUFFER	提供给请求操作的用户缓冲区无效。如果 <i>PhyAddrLen</i> 参数指向的 ULONG 值为零，则会在 Windows Server 2003 及更早版本上返回此错误。
ERROR_NOT_FOUND	找不到元素。如果 <i>SrcIpl</i> 参数未在本地计算机上的接口上指定源 IPv4 地址或 INADDR_ANY IP 地址（IPv4 地址 0.0.0.0），则 Windows Vista 上将返回此错误。
ERROR_NOT_SUPPORTED	在本地计算机上运行的操作系统不支持 SendARP 函数。
其他	如果函数失败，请使用 FormatMessage 获取返回错误的消息字符串。

注解

[SendARP](#) 函数用于请求物理硬件地址（有时称为对应于指定目标 IPv4 地址的 MAC 地址）。如果请求的信息不在本地计算机上的 ARP 表中，则 [SendARP](#) 函数将导致发送 ARP 请

求以获取物理地址。如果函数成功，则会在 *pMacAddr* 参数指向的数组中返回与指定目标 IPv4 地址对应的物理地址。

仅当目标 IPv4 地址位于本地子网上时，IPv4 地址的物理地址才可用，(无需通过任何路由器)即可直接访问该 IPv4 地址。如果目标 IPv4 地址不在本地子网上，则 **SendARP** 函数将失败。

如果 **SendARP** 函数在 Windows Vista 及更高版本上成功，则会使用结果更新本地计算机上的 ARP 表。如果 **SendARP** 函数在 Windows Server 2003 及更早版本上成功，则本地计算机上的 ARP 表不受影响。

Windows Vista 和更高版本上的 **SendARP** 函数返回的错误返回值与 Windows Server 2003 及更早版本上的 **SendARP** 函数不同。

在 Windows Vista 及更高版本中，作为 *pMacAddr* 或 *PhyAddrLen* 参数传递给 **SendARP** 函数的 NULL 指针会导致访问冲突，并且应用程序被终止。如果 Windows Vista 及更高版本发生错误，**并且****ERROR_BAD_NET_NAME**、**ERROR_BUFFER_OVERFLOW** 或 **ERROR_NOT_FOUND** 返回，则 *PhyAddrLen* 参数指向的 ULONG 值将设置为零。如果在 Windows Vista 及更高版本上 *PhyAddrLen* 参数指向的 ULONG 值小于 6，则 **SendARP** 函数将返回**ERROR_BUFFER_OVERFLOW** 指示用于接收物理地址的缓冲区太小。如果 *SrcIp* 参数指定的 IPv4 地址不是本地计算机上的接口，则 Windows Vista 和更高版本上的 **SendARP** 函数将返回 **ERROR_NOT_FOUND**。

在 Windows Server 2003 及更早版本中，作为 *pMacAddr* 或 *PhyAddrLen* 参数传递给 **SendARP** 函数的 NULL 指针将返回**ERROR_INVALID_PARAMETER**。如果在 Windows Server 2003 及更早版本上发生错误，并且返回**ERROR_GEN_FAILURE** 或 **ERROR_INVALID_USER_BUFFER**，则 *PhyAddrLen* 参数指向的 ULONG 值将设置为零。如果在 Windows Server 2003 及更早版本中，*PhyAddrLen* 参数指向的 ULONG 值小于 6，则 **SendARP** 函数不会返回错误，而只返回 *pMacAddr* 参数指向的数组中的部分硬件地址。因此，如果 *PhyAddrLen* 参数指向的值为 4，则在 *pMacAddr* 参数指向的数组中仅返回硬件地址的前 4 个字节。如果 *SrcIp* 参数指定的 IPv4 地址不是本地计算机上的接口，则 Windows Server 2003 及更早版本上的 **SendARP** 函数将忽略 *SrcIp* 参数，并将本地计算机上的 IPv4 地址用于源 IPv4 地址。

[GetIpNetTable](#) 函数检索本地计算机上的 ARP 表，该表将 IPv4 地址映射到物理地址。

[CreateIpNetEntry](#) 函数在本地计算机上的 ARP 表中创建 ARP 条目。

[DeleteIpNetEntry](#) 函数从本地计算机上的 ARP 表中删除 ARP 条目。

[SetIpNetEntry](#) 函数修改本地计算机上 ARP 表中的现有 ARP 条目。

[FlushIpNetTable](#) 函数从本地计算机上的 ARP 表中删除指定接口的所有 ARP 条目。

在 Windows Vista 及更高版本上，[ResolveIpNetEntry2](#) 函数可用于替换 [SendARP](#) 函数。如果传递给 [ResolveIpNetEntry2](#) 函数的 [MIB_IPNET_ROW2](#) 结构的 [Address](#) 成员是 IPv4 地址，则发送 ARP 请求。

在 Windows Vista 上，当传递给这些函数的 [MIB_IPNET_ROW2](#) 结构的 [Address](#) 成员是 IPv4 地址时，可以使用一组新的函数来访问、修改和删除 ARP 表条目。新函数包括：[GetIpNetTable2](#)、[CreateIpNetEntry2](#)、[DeleteIpNetEntry2](#)、[FlushIpNetTable2](#) 和 [SetIpNetEntry2](#)。

有关 [IPAddr](#) 数据类型的信息，请参阅 [Windows 数据类型](#)。若要在点点十进制表示法和 [IPAddr](#) 格式之间转换 IP 地址，请使用 [inet_addr](#) 和 [inet_ntoa](#) 函数。

示例

以下代码演示如何 (与指定的 IPv4 地址关联的 MAC) 地址获取硬件或媒体访问控制。

C++

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "iphlpapi.lib")
#pragma comment(lib, "ws2_32.lib")

void usage(char *pname)
{
    printf("Usage: %s [options] ip-address\n", pname);
    printf("\t -h \t\thelp\n");
    printf("\t -l length \tMAC physical address length to set\n");
    printf("\t -s src-ip \tsource IP address\n");
    exit(1);
}

int __cdecl main(int argc, char **argv)
{
    DWORD dwRetVal;
    IPAddr DestIp = 0;
    IPAddr SrcIp = 0;      /* default for src ip */
    ULONG MacAddr[2];      /* for 6-byte hardware addresses */
    ULONG PhysAddrLen = 6; /* default to length of six bytes */

    char *DestIpString = NULL;
    char *SrcIpString = NULL;
```

```

BYTE *bPhysAddr;
unsigned int i;

if (argc > 1) {
    for (i = 1; i < (unsigned int) argc; i++) {
        if ((argv[i][0] == '-') || (argv[i][0] == '/')) {
            switch (tolower(argv[i][1])) {
            case 'l':
                PhysAddrLen = (ULONG) atol(argv[++i]);
                break;
            case 's':
                SrcIpString = argv[++i];
                SrcIp = inet_addr(SrcIpString);
                break;
            case 'h':
            default:
                usage(argv[0]);
                break;
            }
            /* end switch */
        } else
            DestIpString = argv[i];
    }
    /* end for */
} else
    usage(argv[0]);

if (DestIpString == NULL || DestIpString[0] == '\0')
    usage(argv[0]);

DestIp = inet_addr(DestIpString);

memset(&MacAddr, 0xff, sizeof (MacAddr));

printf("Sending ARP request for IP address: %s\n", DestIpString);

dwRetVal = SendARP(DestIp, SrcIp, &MacAddr, &PhysAddrLen);

if (dwRetVal == NO_ERROR) {
    bPhysAddr = (BYTE *) & MacAddr;
    if (PhysAddrLen) {
        for (i = 0; i < (int) PhysAddrLen; i++) {
            if (i == (PhysAddrLen - 1))
                printf("%.2X\n", (int) bPhysAddr[i]);
            else
                printf("%.2X-", (int) bPhysAddr[i]);
        }
    } else
        printf
            ("Warning: SendArp completed successfully, but returned
length=0\n");
} else {
    printf("Error: SendArp failed with error: %d", dwRetVal);
    switch (dwRetVal) {
    case ERROR_GEN_FAILURE:
        printf(" (ERROR_GEN_FAILURE)\n");
}

```

```

        break;
    case ERROR_INVALID_PARAMETER:
        printf(" (ERROR_INVALID_PARAMETER)\n");
        break;
    case ERROR_INVALID_USER_BUFFER:
        printf(" (ERROR_INVALID_USER_BUFFER)\n");
        break;
    case ERROR_BAD_NET_NAME:
        printf(" (ERROR_GEN_FAILURE)\n");
        break;
    case ERROR_BUFFER_OVERFLOW:
        printf(" (ERROR_BUFFER_OVERFLOW)\n");
        break;
    case ERROR_NOT_FOUND:
        printf(" (ERROR_NOT_FOUND)\n");
        break;
    default:
        printf("\n");
        break;
    }
}

return 0;
}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateIpNetEntry](#)

[CreateIpNetEntry2](#)

[CreateProxyArpEntry](#)

[DeleteIpNetEntry](#)

[DeleteIpNetEntry2](#)

[DeleteProxyArpEntry](#)

[FlushIpNetTable](#)

[FlushIpNetTable2](#)

[GetIpNetEntry2](#)

[GetIpNetTable2](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[IPAddr](#)

[ResolveIpNetEntry2](#)

[SetIpNetEntry](#)

[SetIpNetEntry2](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

setIfEntry 函数 (iphlpapi.h)

项目2023/08/24

SetIfEntry 函数设置接口的管理状态。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD SetIfEntry(
    [in] PMIB_IFROW pIfRow
);
```

parameters

[in] pIfRow

指向 MIB_IFROW 结构的指针。此结构的 dwIndex 成员指定要设置管理状态的接口。dwAdminStatus 成员指定新的管理状态。dwAdminStatus 成员可以是以下值之一。

值	含义
MIB_IF_ADMIN_STATUS_UP	接口在管理上已启用。
MIB_IF_ADMIN_STATUS_DOWN	接口在管理上处于禁用状态。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 及更高版本上返回，包括以下几种情况：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (RunAs 管理员)。
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果找不到由 pIfRow 参数指向的 MIB_IFROW 结构的 dwIndex 成员指定的网络接口，则会在 Windows Vista 及更高版本上返回此错误。

ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>pIfRow</i> 参数中传递 NULL 指针，或者未指定 <i>pIfRow</i> 参数指向 MIB_IFROW 的 dwIndex 成员，则返回此错误。如果找不到由 <i>pIfRow</i> 参数指向的 MIB_IFROW 结构的 dwIndex 成员指定的网络接口，则 Windows Server 2003 及更早版本也会返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果未在本地计算机上配置 TCP/IP 堆栈，则会在 Windows Server 2003 及更早版本上返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

SetIfEntry 函数用于设置本地计算机上接口的管理状态。

pIfRow 参数指向的 MIB_IFROW 结构中的 dwIndex 成员必须初始化为接口索引。

如果 *pIfRow* 参数指向的 MIB_IFROW 的 dwIndex 成员与本地计算机上的现有接口不匹配，则 *SetIfEntry* 函数将失败。

在 Windows Vista 及更高版本上，*SetIfEntry* 函数只能由以 Administrators 组成员身份登录的用户调用。如果 *SetIfEntry* 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。

SetIfEntry 函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。

若要成功执行此函数，调用方必须以 Administrators 组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetIfEntry](#)

[GetIfTable](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IFROW](#)

[MIB_IFTABLE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

setInterfaceDnsSettings 函数 (netioapi.h)

项目2023/04/22

设置 Settings 参数中指定的每个接口 DNS 设置。

语法

C++

```
NETIOAPI_API SetInterfaceDnsSettings(  
    GUID Interface,  
    const DNS_INTERFACE_SETTINGS *Settings  
) ;
```

parameters

Interface

类型: _In_ GUID

设置引用的 COM 接口的 GUID。

Settings

类型: _In_ const DNS_INTERFACE_SETTINGS*

指向包含 DNS 接口设置 的DNS_INTERFACE_SETTINGS类型结构的指针。

如果此参数指向 DNS_INTERFACE_SETTINGS 结构，则必须将
DNS_INTERFACE_SETTINGS: : Version 成员设置为
DNS_INTERFACE_SETTINGS_VERSION1。

如果此参数指向 DNS_INTERFACE_SETTINGS_EX 结构，则必须将版本设置为
DNS_INTERFACE_SETTINGS_VERSION2。

如果此参数指向 DNS_INTERFACE_SETTINGS3 结构，则必须将版本设置为
DNS_INTERFACE_SETTINGS_VERSION3。

必须在 “DNS_INTERFACE_SETTINGS: : Flags” 字段中正确设置所有所需选项，并仅填充为其设置了选项的字段。必须清除没有相应选项的所有其他字段。

返回值

如果成功，则返回NO_ERROR。非零返回值指示失败。

要求

最低受支持的客户端	Windows 10内部版本 19041
最低受支持的服务器	Windows 10内部版本 19041
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

- [GetInterfaceDnsSettings](#)

反馈

此页面是否有帮助？



[在 Microsoft Q&A 获得帮助](#)

SetIpForwardEntry 函数 (iphlpapi.h)

项目2023/08/24

SetIpForwardEntry 函数修改本地计算机的 IPv4 路由表中的现有路由。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD SetIpForwardEntry(
    [in] PMIB_IPFORWARDROW pRoute
);
```

parameters

[in] pRoute

指向 [MIB_IPFORWARDROW](#) 结构的指针，该结构指定现有路由的新信息。调用方必须为此结构的 dwForwardProto 成员指定 [MIB IPPROTO_NETMGMT](#)。调用方还必须为结构的 dwForwardIfIndex、dwForwardDest、dwForwardMask、dwForwardNextHop 和 dwForwardPolicy 成员指定值。

返回值

如果函数成功，则返回值 NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员（运行方式管理员）。
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果找不到由 pRoute 参数指向的 MIB_IPFORWARDROW 结构中的 dwForwardIfIndex 成员指定的网络接口，则会在 Windows Vista 和更高版本上返回此错误。
ERROR_INVALID_PARAMETER	pRoute 参数为 NULL，或者 SetIpForwardEntry 无法从 pRoute 指向的内存中读取，或者 MIB_IPFORWARDROW

	结构的一个成员无效。
ERROR_NOT_FOUND	找不到元素。当为同一 IPv4 路由表条目调用 DeleteIpForwardEntry 函数和 SetIpForwardEntry 函数时，将在 Windows Vista 和更高版本上返回错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果未在本地计算机上配置 IPv4 传输，则返回此值。如果未在本地计算机上配置 TCP/IP 堆栈，则 Windows Server 2003 及更早版本也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

路由参数指向的 [MIB_IPFORWARDROW](#) 结构的 dwForwardProto 成员必须设置为 [MIB IPPROTO_NETMGMT](#) 否则 [SetIpForwardEntry](#) 将失败。路由协议标识符用于标识指定路由协议的路由信息。例如，[MIB IPPROTO_NETMGMT](#) 用于通过网络管理（例如动态主机配置协议 (DHCP)、简单网络管理协议 (SNMP)）或通过调用 [CreateIpForwardEntry](#)、[DeleteIpForwardEntry](#) 或 [SetIpForwardEntry](#) 函数来标识 IP 路由设置的路由信息。

在 Windows Vista 和 Windows Server 2008 上，*pRoute* 参数指向的 [MIB_IPFORWARDROW](#) 结构的 dwForwardMetric1 成员中指定的路由指标表示添加到关联接口的 [MIB_IPINTERFACE_ROW](#) 结构的 **指标** 成员中指定的路由指标的组合。因此，[MIB_IPFORWARDROW](#) 结构的 dwForwardMetric1 成员应等于或大于关联 [MIB_IPINTERFACE_ROW](#) 结构的 Metric 成员。如果应用程序想要将路由指标设置为 0，则应将 [MIB_IPFORWARDROW](#) 结构的 dwForwardMetric1 成员设置为等于关联 [MIB_IPINTERFACE_ROW](#) 结构的 Metric 成员中指定的接口指标的值。应用程序可以通过调用 [GetIpInterfaceEntry](#) 函数来检索接口指标。

在 Windows Vista 和 Windows Server 2008 上，[SetIpForwardEntry](#) 函数仅适用于具有单个子接口的接口（其中接口 LUID 和子接口 LUID 是相同的）。[MIB_IPFORWARDROW](#) 结构的 dwForwardIfIndex 成员指定接口。

[SetIpForwardEntry](#) 当前不使用路由参数指向的 [MIB_IPFORWARDROW](#) 结构 dwForwardAge 成员。仅当路由和远程访问服务 (RRAS) 正在运行时，才使用 dwForwardAge 成员，然后仅对 [协议标识符](#) 引用页上定义的 [MIB IPPROTO_NETMGMT](#) 类型的路由使用。当 dwForwardAge 设置为 INFINITE 时，不会根据超时删除路由

值。dwForwardAge 的任何其他值指定 TCP/IP 堆栈从网络路由表中删除路由之前的秒数。

由 [SetIpForwardEntry](#) 修改的路由将自动具有 DWForwardAge 的默认值 INFINITE。

SetIpForwardEntry 当前不使用路由参数指向的 MIB_IPFORWARDROW 结构中的许多成员。这些成员包括 dwForwardPolicy、dwForwardType、dwForwardAge、dwForwardNextHopAS、dwForwardMetric1、dwForwardMetric2、dwForwardMetric3、dwForwardMetric4 和 dwForwardMetric5。

若要在 IP 路由表中创建新路由，请使用 [CreateIpForwardEntry](#) 函数。若要检索 IP 路由表，请调用 [GetIpForwardTable](#) 函数。

在 Windows Vista 及更高版本上，**SetIpForwardEntry** 函数只能由以管理员组成员身份登录的用户调用。如果 **SetIpForwardEntry** 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。

此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

示例

以下示例演示如何将默认网关更改为 NewGateway。只需调用 [GetIpForwardTable](#)，更改网关，然后调用 **SetIpForwardEntry** 不会更改路由，而只会添加新路由。如果由于某种原因存在多个默认网关，此代码将删除它们。请注意，新网关必须可行；否则，TCP/IP 将忽略更改。

注意 执行此代码将更改 IP 路由表，并可能导致网络活动失败。

Windows Vista 及更高版本：当为 Windows Vista 和更高版本的相同路由表条目调用 [DeleteIpForwardEntry](#) 函数和 **SetIpForwardEntry** 函数时，将返回 ERROR_NOT_FOUND。在 Windows Vista 及更高版本上复制此示例的正确方法是使用 [CreateIpForwardEntry](#) 函数创建新的路由表条目，然后通过调用 [DeleteIpForwardEntry](#) 函数删除旧的路由表条目。

C++

```
#pragma comment(lib, "IPHLPAPI.lib")

// #ifndef WIN32_LEAN_AND_MEAN
// #define WIN32_LEAN_AND_MEAN
// #endif

// #pragma warning(push)
// #pragma warning(disable: 4127)

// #include <windows.h>

#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <stdio.h>

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))
/* Note: could also use malloc() and free() */

int main()
{
    // Declare and initialize variables.

    /* variables used for SetIfForwardEntry */

    PMIB_IPFORWARDTABLE pIpForwardTable = NULL;
    PMIB_IPFORWARDROW pRow = NULL;
    DWORD dwSize = 0;
    BOOL bOrder = FALSE;
    DWORD dwStatus = 0;
    DWORD NewGateway = 0xDDBBCCAA;           // this is in host order Ip Address
    AA.BB.CC.DD is DDCCBAA
    DWORD i;

    // Find out how big our buffer needs to be.
    dwStatus = GetIpForwardTable(pIpForwardTable, &dwSize, bOrder);
    if (dwStatus == ERROR_INSUFFICIENT_BUFFER) {
        // Allocate the memory for the table
        pIpForwardTable = (PMIB_IPFORWARDTABLE) malloc(dwSize);
        if (pIpForwardTable == NULL) {
            printf("Unable to allocate memory for the IPFORWARDTALE\n");
            exit(1);
        }
        // Now get the table.
        dwStatus = GetIpForwardTable(pIpForwardTable, &dwSize, bOrder);
    }

    if (dwStatus != ERROR_SUCCESS) {
        printf("getIpForwardTable failed.\n");
        if (pIpForwardTable)
```

```

        free(pIpForwardTable);
        exit(1);
    }
    // Search for the row in the table we want. The default gateway has a
    destination
    // of 0.0.0.0. Notice that we continue looking through the table, but copy
    only
    // one row. This is so that if there happen to be multiple default gateways,
    we can
    // be sure to delete them all.
    for (i = 0; i < pIpForwardTable->dwNumEntries; i++) {
        if (pIpForwardTable->table[i].dwForwardDest == 0) {
            // We have found the default gateway.
            if (!pRow) {
                // Allocate some memory to store the row in. This is easier
                than filling
                // in the row structure ourselves, and we can be sure to
                change only the
                // gateway address.
                pRow = (PMIB_IPFORWARDROW) malloc(sizeof
                (MIB_IPFORWARDROW));
                if (!pRow) {
                    printf("Malloc failed. Out of memory.\n");
                    exit(1);
                }
                // Copy the row.
                memcpy(pRow, &(pIpForwardTable->table[i]),
                sizeof (MIB_IPFORWARDROW));
            }
            // Delete the old default gateway entry.
            dwStatus = DeleteIpForwardEntry(&(pIpForwardTable->table[i]));

            if (dwStatus != ERROR_SUCCESS) {
                printf("Could not delete old gateway\n");
                exit(1);
            }
        }
    }

    // Set the nexthop field to our new gateway. All the other properties of the
    route will
    // be the same as they were previously.
    pRow->dwForwardNextHop = NewGateway;

    // Create a new route entry for the default gateway.
    dwStatus = SetIpForwardEntry(pRow);

    if (dwStatus == NO_ERROR)
        printf("Gateway changed successfully\n");
    else if (dwStatus == ERROR_INVALID_PARAMETER)
        printf("Invalid parameter.\n");
    else
        printf("Error: %d\n", dwStatus);

    // Free resources.

```

```
    if (pIpForwardTable)
        free(pIpForwardTable);
    if (pRow)
        free(pRow);
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[CreateIpForwardEntry](#)

[DeleteIpForwardEntry](#)

[GetIpForwardTable](#)

[GetIpInterfaceEntry](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPFORWARDROW](#)

[MIB_IPINTERFACE_ROW](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

SetIpForwardEntry2 函数 (netioapi.h)

项目2023/08/25

SetIpForwardEntry2 函数设置本地计算机上的 IP 路由条目的属性。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API SetIpForwardEntry2(
    [in] const MIB_IPFORWARD_ROW2 *Route
);
```

parameters

[in] Route

指向 IP 路由条目 [MIB_IPFORWARD_ROW2](#) 结构条目的指针。必须将 [MIB_IPFORWARD_ROW2](#) 的 DestinationPrefix 成员设置为有效的 IP 目标前缀，必须将 [MIB_IPFORWARD_ROW2](#) 的 NextHop 成员设置为有效的 IP 地址系列和 IP 地址，并且必须指定 [MIB_IPFORWARD_ROW2](#) 的 InterfaceLuid 或 InterfaceIndex 成员。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置的管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>Route</i> 参数中传递 NULL 指针，未指定 <i>Route</i> 参数所指向 MIB_IPFORWARD_ROW2 的 DestinationPrefix 成员，未指定 <i>Route</i> 参数指向的 MIB_IPFORWARD_ROW2 的 NextHop 成员，或者 <i>Route</i> 参数所指向 MIB_IPFORWARD_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员，则返回此错误未指定参数。

ERROR_NOT_FOUND	找不到指定的接口。如果找不到 <i>Route</i> 参数指向的 MIB_IPFORWARD_ROW2 的 <i>InterfaceLuid</i> 或 <i>InterfaceIndex</i> 成员指定的网络接口，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`SetIpForwardEntry2` 函数在 Windows Vista 及更高版本上定义。

`SetIpForwardEntry2` 函数用于设置本地计算机上现有 IP 路由条目的属性。

Route 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构中的 *DestinationPrefix* 成员必须初始化为有效的 IP 地址前缀和系列。*Route* 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构中的 *NextHop* 成员必须初始化为有效的 IP 地址和系列。此外，指向 *Route* 参数的 [MIB_IPFORWARD_ROW2](#) 结构中至少有一个成员必须初始化到接口：*InterfaceLuid* 或 *InterfaceIndex*。

字段按上面列出的顺序使用。因此，如果指定了 *InterfaceLuid*，则此成员用于确定要添加单播 IP 地址的接口。如果未为 *InterfaceLuid* 成员设置值（此成员的值设置为零），则接下来使用 *InterfaceIndex* 成员来确定接口。

Route 参数指向的 [MIB_IPFORWARD_ROW2](#) 结构的 *Metric* 成员中指定的路由指标偏移量仅代表完整路由指标的一部分。完整指标是添加到关联接口的[MIB_IPINTERFACE_ROW](#)结构的指标成员中指定的接口指标的路由指标偏移量的组合。应用程序可以通过调用 [GetIpInterfaceEntry](#) 函数来检索接口指标。

调用 `SetIpForwardEntry2` 函数时，将忽略行指向的 [MIB_IPFORWARD_ROW2](#) 结构的 *Age* 和 *Origin* 成员。这些成员由网络堆栈设置，无法使用 `SetIpForwardEntry2` 函数进行更改。

如果 *Route* 参数指向的[MIB_IPFORWARD_ROW2](#) 的 *DestinationPrefix* 和 *NextHop* 成员与指定接口上的 IP 路由条目不匹配，`SetIpForwardEntry2` 函数将失败。

`SetIpForwardEntry2` 函数只能由以 Administrators 组成员身份登录的用户调用。如果 `SetIpForwardEntry2` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。

`SetIpForwardEntry2` 函数也可能因为用户帐户控制（Windows Vista 及更高版本上的 UAC）而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 *requestedExecutionLevel* 设置为 *requireAdministrator*。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员（RunAs 管理员）此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIphForwardEntry2](#)

[DeleteIphForwardEntry2](#)

[GetBestRoute2](#)

[GetIphForwardEntry2](#)

[GetIphForwardTable2](#)

[GetIplInterfaceEntry](#)

[InitializeIphForwardEntry](#)

[MIB_IPFORWARD_ROW2](#)

[MIB_IPFORWARD_TABLE2](#)

[MIB_IPINTERFACE_ROW](#)

[NotifyRouteChange2](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

SetIpInterfaceEntry 函数 (netioapi.h)

项目2023/08/25

SetIpInterfaceEntry 函数设置本地计算机上的 IP 接口的属性。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API SetIpInterfaceEntry(
    [in, out] PMIB_IPINTERFACE_ROW Row
);
```

parameters

[in, out] Row

指向接口 [MIB_IPINTERFACE_ROW](#) 结构条目的指针。 输入时，必须将 [MIB_IPINTERFACE_ROW](#) 的 Family 成员设置为 AF_INET6 或 AF_INET，并且必须指定 [MIB_IPINTERFACE_ROW](#) 的 InterfaceLuid 或 InterfaceIndex 成员。 成功返回时，如果指定了 [MIB_IPINTERFACE_ROW](#) 项的 InterfaceIndex 成员，则会填充 [MIB_IPINTERFACE_ROW](#) 的 InterfaceLuid 成员。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_FILE_NOT_FOUND	系统找不到指定的文件。如果由 Row 参数指向的 MIB_IPINTERFACE_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口 LUID 或接口索引不是本地计算机上的值，则返回此错误。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数所指向 MIB_IPINTERFACE_ROW 的 Family

	成员未指定为 AF_INET 或 AF_INET6，或者 Row 参数指向的 MIB_IPINTERFACE_ROW 的 InterfaceLuid 或 InterfaceIndex 成员均未指定，则返回此错误。
ERROR_NOT_FOUND	找不到指定的接口。如果由 Row 参数指向的 MIB_IPINTERFACE_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口与 MIB_IPINTERFACE_ROW 结构中的 Family 成员中指定的 IP 地址系列不匹配，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

SetIpInterfaceEntry 函数在 Windows Vista 及更高版本上定义。

SetIpInterfaceEntry 函数可用于修改现有 IP 接口条目。

输入时，Row 参数指向的 MIB_IPINTERFACE_ROW 结构中的 Family 成员必须初始化为 AF_INET 或 AF_INET6。此外，在输入时，必须初始化指向 Row 参数的 MIB_IPINTERFACE_ROW 结构中的以下至少一个成员：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则使用此成员来确定接口。如果没有为 InterfaceLuid 成员设置值，(此成员的值) 设置为零，则接下来使用 InterfaceIndex 成员来确定接口。

在输出时，如果指定了 InterfaceIndex，则填充 Row 参数指向的 MIB_IPINTERFACE_ROW 结构的 InterfaceLuid 成员。

调用 SetIpInterfaceEntry 函数时，将忽略行指向的 MIB_IPINTERFACE_ROW 结构的 MaxReassemblySize、MinRouterAdvertisementInterval、MaxRouterAdvertisementInterval、Connected、SupportsWakeUpPatterns、SupportsNeighborDiscovery、SupportsRouterDiscovery、ReachableTime、TransmitOffload 和 ReceiveOffload 成员。这些成员由网络堆栈设置，不能使用 SetIpInterfaceEntry 函数进行更改。

应用程序通常会调用 GetIpInterfaceTable 函数来检索本地计算机上的 IP 接口条目，或调用 GetIpInterfaceEntry 函数来仅检索要修改的 IP 接口条目。然后，可以修改特定 IP 接口条目的 MIB_IPINTERFACE_ROW 结构，并将指向此结构的指针传递给 Row 参数中的 SetIpInterfaceEntry 函数。但是，对于 IPv4，应用程序不得尝试修改 MIB_IPINTERFACE_ROW 结构的 SitePrefixLength 成员。对于 IPv4，SitePrefixLength 成员必须设置为 0。

修改现有 IP 接口条目的另一种可能方法是使用 InitializeIpInterfaceEntry 函数使用默认值初始化 MIB_IPINTERFACE_ROW 结构条目的字段。然后，在 Row 参数指向的

`MIB_IPINTERFACE_ROW` 结构中设置 `Family` 成员和 `InterfaceIndex` 或 `InterfaceLuid` 成员，以匹配要更改的 IP 接口。然后，应用程序可以更改要修改 的 `MIB_IPINTERFACE_ROW` 条目中的字段，然后调用 `SetIpInterfaceEntry` 函数。但是，对于 IPv4，应用程序不得尝试修改`MIB_IPINTERFACE_ROW`结构的 `SitePrefixLength` 成员。对于 IPv4，`SitePrefixLength` 成员必须设置为 0。必须谨慎使用此方法，因为确定要更改的所有字段的唯一 `MIB_IPINTERFACE_ROW` 方法是，在初始化为默认值时，将特定 IP 接口条目`MIB_IPINTERFACE_ROW`中的字段与 `InitializeIpInterfaceEntry` 函数设置的字段进行比较。

对具有不同安全要求的多个网络的无特权同时访问会产生安全漏洞，并允许无特权应用程序意外地在两个网络之间中继数据。一个典型示例是同时访问虚拟专用网络 (VPN) 和 Internet。Windows Server 2003 和 Windows XP 使用弱主机模型，其中 RAS 通过增加其他接口上所有默认路由的路由指标来阻止此类同时访问。因此，所有流量都通过 VPN 接口路由，从而中断其他网络连接。

在 Windows Vista 及更高版本上，默认使用强主机模型。如果使用 `GetBestRoute2` 或 `GetBestRoute` 在路由查找中指定了源 IP 地址，则路由查找仅限于源 IP 地址的接口。RAS 的路由指标修改不起作用，因为潜在路由列表甚至没有 VPN 接口的路由，从而允许流量流向 Internet。`MIB_IPINTERFACE_ROW` 的 `DisableDefaultRoutes` 成员可用于在接口上使用默认路由禁用。VPN 客户端可以将此成员用作安全措施，以便在 VPN 客户端不需要拆分隧道时限制拆分隧道。VPN 客户端可以调用 `SetIpInterfaceEntry` 函数，以便在需要时将 `DisableDefaultRoutes` 成员设置为 TRUE。VPN 客户端可以通过调用 `GetIpInterfaceEntry` 函数来查询 `DisableDefaultRoutes` 成员的当前状态。

The

`SetIpInterfaceEntry` 函数只能由以管理员组成员身份登录的用户调用。如果 `SetIpInterfaceEntry` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。此函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]

目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[GetBestRoute](#)

[GetBestRoute2](#)

[GetIfEntry2](#)

[GetIfTable2](#)

[GetIfTable2Ex](#)

[GetIplInterfaceEntry](#)

[GetIplInterfaceTable](#)

[IP 帮助程序函数参考](#)

[InitializeIplInterfaceEntry](#)

[MIB_IF_ROW2](#)

[MIB_IF_TABLE2](#)

[MIB_IPINTERFACE_ROW](#)

[MIB_IPINTERFACE_TABLE](#)

[NotifyIplInterfaceChange](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

SetIpNetEntry 函数 (iphlpapi.h)

项目2023/08/24

SetIpNetEntry 函数修改本地计算机上的 ARP 表中的现有 ARP 条目。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD SetIpNetEntry(
    [in] PMIB_IPNETROW pArpEntry
);
```

parameters

[in] pArpEntry

指向 MIB_IPNETROW 结构的指针。此结构中的信息指定要修改的条目和条目的新信息。调用方必须为此结构的所有成员指定值。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员（运行方式管理员）。
ERROR_INVALID_PARAMETER	pArpEntry 参数为 NULL，或者 SetIpNetEntry 无法从 pArpEntry 指向的内存中读取，或者 MIB_IPNETROW 结构的一个成员无效。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

在 Windows Vista 及更高版本中，`SetIpNetEntry` 函数只能由以管理员组成员身份登录的用户调用。如果 `SetIpNetEntry` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。

`SetIpNetEntry` 函数也可能因为用户帐户控制 (Windows Vista 及更高版本上的 UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录 (而非内置管理员) 的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果应用程序缺少此清单文件，则作为管理员组成员 (而非内置管理员) 登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。

若要成功执行此函数，调用方必须以管理员组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

[CreateIpNetEntry](#)

[DeleteIpNetEntry](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

SetIpNetEntry2 函数 (netioapi.h)

项目2023/08/25

SetIpNetEntry2 函数设置本地计算机上现有邻居 IP 地址条目的物理地址。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API SetIpNetEntry2(
    [in, out] PMIB_IPNET_ROW2 Row
);
```

parameters

[in, out] Row

指向邻居 IP 地址条目 [MIB_IPNET_ROW2](#) 结构条目的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数指向 MIB_IPNET_ROW2 的 Address 成员未设置为有效的单播、任意广播或多播 IPv4 或 IPv6 地址，Row 参数指向的 MIB_IPNET_ROW2 的 PhysicalAddress 和 PhysicalAddressLength 成员未设置为有效的物理地址，则返回此错误。或 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员均未指定。如果在 Address 成员中传递了环回地址，也会返回此错误。
ERROR_NOT_FOUND	找不到指定的接口。如果找不到 Row 参数指向的 MIB_IPNET_ROW2 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口，则返回此错误。

ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且 Row 参数指向的 MIB_IPNET_ROW2 的 Address 成员中指定了 IPv4 地址，或者本地计算机上没有 IPv6 堆栈，并且地址成员中指定了 IPv6 地址，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

`SetIpNetEntry2` 函数在 Windows Vista 及更高版本上定义。

`SetIpNetEntry2` 函数用于设置本地计算机上现有邻居 IP 地址条目的物理地址。

Row 参数指向的 MIB_IPNET_ROW2 结构中的 Address 成员必须初始化为有效的单播、任意广播或多播 IPv4 或 IPv6 地址和系列。Row 参数指向的 MIB_IPNET_ROW2 结构中的 PhysicalAddress 和 PhysicalAddressLength 成员必须初始化为有效的物理地址。此外，必须将指向 Row 参数的 MIB_IPNET_ROW2 结构中的以下成员中的至少一个初始化为接口：InterfaceLuid 或 InterfaceIndex。

字段按上面列出的顺序使用。因此，如果指定了 InterfaceLuid，则此成员用于确定要添加单播 IP 地址的接口。如果没有为 InterfaceLuid 成员设置值，(此成员的值) 设置为零，则接下来使用 InterfaceIndex 成员来确定接口。

如果 Row 参数指向的 MIB_IPNET_ROW2 的 Address 成员中传递的 IP 地址不是指定接口上的现有邻居 IP 地址，则 `SetIpNetEntry2` 函数将失败。

`SetIpNetEntry2` 函数只能由以管理员组成员身份登录的用户调用。如果 `SetIpNetEntry2` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。

`SetIpNetEntry2` 函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

要求

最低受支持的客户端

Windows Vista [仅限桌面应用]

最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateIpNetEntry2](#)

[DeleteIpNetEntry2](#)

[FlushIpNetTable2](#)

[GetIpNetEntry2](#)

[GetIpNetTable2](#)

[MIB_IPNET_ROW2](#)

[MIB_IPNET_TABLE2](#)

[ResolveIpNetEntry2](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

SetIpStatistics 函数 (iphlpapi.h)

项目2023/08/24

SetIpStatistics 函数打开或关闭 IP 转发，并设置本地计算机的默认生存时间 (TTL) 值。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD SetIpStatistics(
    [in] PMIB_IPSTATS pIpStats
);
```

parameters

[in] pIpStats

指向 MIB_IPSTATS 结构的指针。调用方应将此结构的 dwForwarding 和 dwDefaultTTL 成员设置为新值。若要使其中一个成员保持其当前值，请使用 MIB_USE_CURRENT_TTL 或 MIB_USE_CURRENT_FORWARDING。

返回值

如果函数成功，则返回值 NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>pIpStats</i> 参数中传递 NULL 指针，则返回此错误。如果 <i>pIpStats</i> 参数指向的 MIB_IPSTATS 结构中的 dwForwarding 成员包含除 MIB_IP_NOT_FORWARDING、MIB_IP_FORWARDING 或 MIB_USE_CURRENT_FORWARDING 以外的值，也会返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

若要仅设置默认 TTL，调用方也可以使用 [SetIpTTL](#) 函数。

在 Windows Vista 及更高版本中，[SetIpStatistics](#) 函数只能由以 Administrators 组成员身份登录的用户调用。如果 [SetIpStatistics](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。

[SetIpStatistics](#) 函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。

若要成功执行此函数，调用方必须以 Administrators 组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPSTATS](#)

[SetIpStatisticsEx](#)

[SetIpTTL](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

SetIpStatisticsEx 函数 (iphlpapi.h)

项目2023/08/24

SetIpStatisticsEx 函数打开或关闭 IP 转发，并设置本地计算机的默认生存时间 (TTL) 值。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG SetIpStatisticsEx(
    [in] PMIB_IPSTATS Statistics,
    ULONG           Family
);
```

parameters

[in] Statistics

指向 MIB_IPSTATS 结构的指针。调用方应将此结构的 dwForwarding 和 dwDefaultTTL 成员设置为新值。若要使其中一个成员保持其当前值，请使用 MIB_USE_CURRENT_TTL 或 MIB_USE_CURRENT_FORWARDING。

Family

要为其设置转发和 TTL 的地址系列。

Winsock2.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和 PF_ 协议系列常量的值 (相同，例如，AF_INET 和 PF_INET)，因此可以使用任一常量。

在针对 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织方式已更改，并且 Ws2def.h 头文件中定义了此成员的可能值。请注意，Ws2def.h 头文件会自动包含在 Winsock2.h 中，永远不应直接使用。

当前支持的值 AF_INET, AF_INET6。

值	含义
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。指定此参数后，此函数将设置 IPv4 目的转发和 TTL 选项。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。指定此参数时，此函数将设置 IPv6 目的转发和 TTL 选项。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 <i>plpStats</i> 参数中传递 NULL 指针，或者 <i>Family</i> 参数未设置为 AF_INET，则返回此错误，AF_INET6。如果 <i>plpStats</i> 参数指向的 MIB_IPSTATS 结构中的 <i>dwForwarding</i> 成员包含除 MIB_IP_NOT_FORWARDING、MIB_IP_FORWARDING 或 MIB_USE_CURRENT_FORWARDING 以外的值，也会返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，并且已在 <i>Family</i> 参数中指定了 AF_INET，或者本地计算机上没有 IPv6 堆栈，并且 AF_INET6 已在 <i>Family</i> 成员中指定，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

若要仅设置默认 TTL，调用方也可以使用 [SetIpTTL](#) 函数。

[SetIpStatisticsEx](#) 函数只能由以管理员组成员身份登录的用户调用。如果 [SetIpStatisticsEx](#) 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。

[SetIpStatisticsEx](#) 函数也可能因为用户帐户控制 (Windows Vista 及更高版本上的 UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录 (而非内置管理员) 的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果上的应用程序缺少此清单文件，则以管理员组成员身份登录的用户 (而不是内置管理员) 必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员)，此函数才能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPSTATS](#)

[SetIpStatistics](#)

[SetIpTTL](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

SetIpTTL 函数 (iphlpapi.h)

项目2023/08/24

SetIpTTL 函数设置本地计算机的默认生存时间 (TTL) 值。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD SetIpTTL(
    [in] UINT nTTL
);
```

parameters

[in] nTTL

本地计算机的新 TTL 值。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果 nTTL 参数无效，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

也可以使用 [SetIpStatistics](#) 函数设置默认 TTL。

在 Windows Vista 及更高版本上，`SetIpTTL` 函数只能由以 Administrators 组成员身份登录的用户调用。如果 `SetIpTTL` 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 `ERROR_ACCESS_DENIED`。

`SetIpStatistics` 函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

注意 在 Windows NT 4.0 和 Windows 2000 及更高版本上，此函数执行特权操作。若要成功执行此函数，调用方必须以 Administrators 组或 NetworkConfigurationOperators 组的成员身份登录。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_IPSTATS](#)

[SetIpStatistics](#)

[SetIpStatisticsEx](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

SetPerTcp6ConnectionEStats 函数 (iphlpapi.h)

项目2023/08/24

SetPerTcp6ConnectionEStats 函数在 IPv6 TCP 连接的读/写信息中设置值。此函数用于启用或禁用 IPv6 TCP 连接的扩展统计信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG SetPerTcp6ConnectionEStats(
    PMIB_TCP6ROW      Row,
    TCP_ESTATS_TYPE   EstatsType,
    PUCHAR            Rw,
    ULONG              RwVersion,
    ULONG              RwSize,
    ULONG              Offset
);
```

parameters

Row

指向 IPv6 TCP 连接的 [MIB_TCP6ROW](#) 结构的指针。

EstatsType

要设置的 TCP 扩展统计信息的类型。此参数确定 *Rw* 参数中预期的数据和信息格式。

此参数可以是 *Tcpestats.h* 头文件中定义的 [TCP_ESTATS_TYPE](#) 枚举类型的值之一。

值	含义
TcpConnectionEstatsData	此值指定 TCP 连接的扩展数据传输信息。 指定此值时， <i>Rw</i> 参数指向的缓冲区应指向 TCP_ESTATS_DATA_RW_v0 结构。
TcpConnectionEstatsSndCong	此值指定 TCP 连接的发送方拥塞。 指定此值时， <i>Rw</i> 参数指向的缓冲区应指向 TCP_ESTATS SND CONG RW v0 结构。
TcpConnectionEstatsPath	此值指定 TCP 连接的扩展路径度量信息。

	指定此值时， <i>Rw</i> 参数指向的缓冲区应指向 TCP_ESTATS_PATH_RW_v0 结构。
TcpConnectionEstatsSendBuff	此值指定 TCP 连接的扩展输出队列信息。 指定此值时， <i>Rw</i> 参数指向的缓冲区应指向 TCP_ESTATS_SEND_BUFF_RW_v0 结构。
TcpConnectionEstatsRec	此值指定 TCP 连接的扩展本地接收器信息。 指定此值时， <i>Rw</i> 参数指向的缓冲区应指向 TCP_ESTATS_REC_RW_v0 结构。
TcpConnectionEstatsObsRec	此值指定 TCP 连接的扩展远程接收器信息。 指定此值时， <i>Rw</i> 参数指向的缓冲区应指向 TCP_ESTATS_OBS_REC_RW_v0 结构。
TcpConnectionEstatsBandwidth	此值指定带宽上的 TCP 连接的带宽估计统计信息。 指定此值时， <i>Rw</i> 参数指向的缓冲区应指向 TCP_ESTATS_BANDWIDTH_RW_v0 结构。
TcpConnectionEstatsFineRtt	此值指定 TCP 连接的精细往返时间 (RTT) 估计统计信息。 指定此值时， <i>Rw</i> 参数指向的缓冲区应指向 TCP_ESTATS_FINE_RTT_RW_v0 结构。

Rw

指向缓冲区的指针，该缓冲区包含要设置的读/写信息。 缓冲区应包含每个结构成员 [的 TCP_BOOLEAN_OPTIONAL 枚举](#) 中的值，该值指定每个成员应如何更新。

RwVersion

要设置的读/写信息的版本。 对于 Windows Vista、Windows Server 2008 和 Windows 7，此参数应设置为零。

RwSize

Rw 参数指向的缓冲区的大小（以字节为单位）。

Offset

要设置的 *Rw* 参数指向结构中成员的偏移量（以字节为单位）。 此参数当前未使用，必须设置为零。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	参数不正确。如果 <i>Row</i> 参数为 NULL 指针，则返回此错误。
ERROR_INVALID_USER_BUFFER	提供给请求操作的用户缓冲区无效。如果 <i>Row</i> 参数为 NULL 指针且 <i>RwSize</i> 参数为非零值，则返回此错误。
ERROR_NOT_FOUND	找不到此请求的条目。如果找不到 <i>Row</i> 参数中指定的 TCP 连接，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果未将 <i>RwVersion</i> 或 <i>Offset</i> 参数设置为 0，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

SetPerTcp6ConnectionEStats 函数是在 Windows Vista 及更高版本上定义的。

SetPerTcp6ConnectionEStats 函数用于为 *Row* 参数中传递的 IPv6 TCP 连接启用或禁用扩展统计信息。默认情况下，TCP 连接的扩展统计信息处于禁用状态。

SetPerTcp6ConnectionEStats 函数用于在 IPv6 TCP 连接的扩展统计信息的读/写信息中设置成员的值。要设置的结构的类型和格式由 *EstatsType* 参数指定。*Rw* 参数包含指向所传递结构的指针。要在此结构中设置的成员由 *Offset* 参数指定。必须指定 *Rw* 参数指向的结构中的所有成员。

目前支持的唯一 TCP 连接统计信息版本是版本零。因此，传递给 **SetPerTcp6ConnectionEStats** 的 *RwVersion* 参数应设置为 0。

传递此函数的 *Rw* 参数指向的结构取决于在 *EstatsType* 参数中传递的枚举值。下表指示应为每个可能的 *EstatsType* 参数类型在 *Rw* 参数中传递的结构类型。

<i>EstatsType</i>	<i>Rw</i> 指向的结构
TcpConnectionEstatsData	TCP_ESTATS_DATA_RW_v0
TcpConnectionEstatsSndCong	TCP_ESTATS SND CONG RW_v0
TcpConnectionEstatsPath	TCP_ESTATS_PATH_RW_v0
TcpConnectionEstatsSendBuff	TCP_ESTATS_SEND_BUFF_RW_v0

TcpConnectionEstatsRec	TCP_ESTATS_REC_RW_v0
TcpConnectionEstatsObsRec	TCP_ESTATS_OBS_REC_RW_v0
TcpConnectionEstatsBandwidth	TCP_ESTATS_BANDWIDTH_RW_v0
TcpConnectionEstatsFineRtt	TCP_ESTATS_FINE_RTT_RW_v0

Offset 参数当前未使用。 *Rw* 参数指向的可能结构都具有单个成员，
TCP_ESTATS_BANDWIDTH_RW_v0 结构除外。当 *EstatsType* 参数设置为
TcpConnectionEstatsBandwidth 时，*Rw* 参数指向的 **TCP_ESTATS_BANDWIDTH_RW_v0**
结构必须在对 **SetPerTcp6ConnectionEStats** 函数的单个调用中同时将两个结构成员都设
置为首选值。

如果 *RwSize* 参数设置为 0，则 **SetPerTcp6ConnectionEStats** 函数将返回 **NO_ERROR** 且
不会更改扩展统计信息状态。

GetTcp6Table 函数用于检索本地计算机上的 IPv6 TCP 连接表。此函数返回包含
MIB_TCP6ROW 项数组的 **MIB_TCP6TABLE** 结构。传递给 **SetPerTcp6ConnectionEStats**
函数的 *Row* 参数必须是现有 IPv6 TCP 连接的条目。

在 IPv6 的 TCP 连接上启用扩展统计信息后，应用程序将调用
GetPerTcp6ConnectionEStats 函数来检索 TCP 连接上的扩展统计信息。

GetPerTcp6ConnectionEStats 函数旨在使用 TCP 诊断网络和应用程序中的性能问题。如
果基于网络的应用程序性能不佳，TCP 可以确定瓶颈是在发送方、接收方还是网络本身。
如果瓶颈在网络中，TCP 可以提供有关其性质的特定信息。

有关 IPv4 连接上的扩展 TCP 统计信息的信息，请参阅 **GetPerTcpConnectionEStats** 和
SetPerTcpConnectionEStats 函数。

SetPerTcp6ConnectionEStats 函数只能由以 Administrators 组成员身份登录的用户调
用。如果 **SetPerTcp6ConnectionEStats** 由不是 Administrators 组成员的用户调用，则函
数调用将失败并返回 **ERROR_ACCESS_DENIED**。此函数也可能因为 Windows Vista 和
Windows Server 2008 上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由
以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用
程序已在清单文件中标记为 **requestedExecutionLevel** 设置为 **requireAdministrator**。如
果 Windows Vista 或 Windows Server 2008 上的应用程序缺少此清单文件，则作为管
理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内
置管理员 (RunAs 管理员) 才能使此功能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[GetTcp6Table](#)

[MIB_TCP6ROW](#)

[MIB_TCP6TABLE](#)

[SetPerTcpConnectionEStats](#)

[TCP_BOOLEAN_OPTIONAL](#)

[TCP_ESTATS_BANDWIDTH_RW_v0](#)

[TCP_ESTATS_DATA_RW_v0](#)

[TCP_ESTATS_FINE_RTT_RW_v0](#)

[TCP_ESTATS_OBS_REC_RW_v0](#)

[TCP_ESTATS_PATH_RW_v0](#)

[TCP_ESTATS_REC_RW_v0](#)

[TCP_ESTATS_SEND_BUFF_RW_v0](#)

[TCP_ESTATS SND CONG RW_v0](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

SetPerTcpConnectionEStats 函数 (iphlpapi.h)

项目2023/08/24

SetPerTcpConnectionEStats 函数在 IPv4 TCP 连接的读/写信息中设置值。此函数用于启用或禁用 IPv4 TCP 连接的扩展统计信息。

语法

C++

```
IPHLPAPI_DLL_LINKAGE ULONG SetPerTcpConnectionEStats(
    PMIB_TCPROW      Row,
    TCP_ESTATS_TYPE  EstatsType,
    PUCHAR          Rw,
    ULONG           RwVersion,
    ULONG           RwSize,
    ULONG           Offset
);
```

parameters

Row

指向 IPv4 TCP 连接的 [MIB_TCPROW](#) 结构的指针。

EstatsType

要设置的 TCP 扩展统计信息的类型。此参数确定 *Rw* 参数中预期的数据和信息格式。

此参数可以是 *Tcpestats.h* 头文件中定义的 [TCP_ESTATS_TYPE](#) 枚举类型的值之一。

Rw

指向缓冲区的指针，该缓冲区包含要设置的读/写信息。缓冲区应包含每个结构成员 [的 TCP_BOOLEAN_OPTIONAL](#) 枚举中的值，该值指定每个成员应如何更新。

RwVersion

要设置的读/写信息的版本。对于 Windows Vista、Windows Server 2008 和 Windows 7，此参数应设置为零。

RwSize

Rw 参数指向的缓冲区的大小（以字节为单位）。

Offset

要设置的 *Rw* 参数指向结构中成员的偏移量（以字节为单位）。此参数当前未使用，必须设置为零。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_USER_BUFFER	提供给请求操作的用户缓冲区无效。如果 <i>Row</i> 参数为 NULL 指针且 <i>RwSize</i> 参数为非零值，则返回此错误。
ERROR_INVALID_PARAMETER	参数不正确。如果 <i>Row</i> 参数为 NULL 指针，则返回此错误。
ERROR_NOT_FOUND	找不到此请求的条目。如果找不到 <i>Row</i> 参数中指定的 TCP 连接，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果未将 <i>RwVersion</i> 或 <i>Offset</i> 参数设置为 0，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

SetPerTcpConnectionEStats 函数在 Windows Vista 及更高版本上定义。

SetPerTcpConnectionEStats 函数用于在 *Row* 参数中传递的 IPv4 TCP 连接上启用或禁用扩展统计信息。默认情况下，TCP 连接的扩展统计信息处于禁用状态。

SetPerTcpConnectionEStats 函数用于在 IPv4 TCP 连接的扩展统计信息的读/写信息中设置成员的值。要设置的结构的类型和格式由 *EstatsType* 参数指定。*Rw* 参数包含指向所传递结构的指针。必须指定 *Rw* 参数指向的结构中的所有成员。

目前支持的唯一 TCP 连接统计信息版本是版本零。因此，传递给 **SetPerTcpConnectionEStats** 的 *RwVersion* 参数应设置为 0。

传递此函数的 *Rw* 参数指向的结构取决于 在 *EstatsType* 参数中传递的枚举值。下表指示应为每个可能的 *EstatsType* 参数类型在 *Rw* 参数中传递的结构类型。

<i>EstatsType</i>	<i>Rw</i> 指向的结构
TcpConnectionEstatsData	TCP_ESTATS_DATA_RW_v0
TcpConnectionEstatsSndCong	TCP_ESTATS SND CONG RW_v0
TcpConnectionEstatsPath	TCP_ESTATS_PATH_RW_v0
TcpConnectionEstatsSendBuff	TCP_ESTATS_SEND_BUFF_RW_v0
TcpConnectionEstatsRec	TCP_ESTATS_REC_RW_v0
TcpConnectionEstatsObsRec	TCP_ESTATS_OBS_REC_RW_v0
TcpConnectionEstatsBandwidth	TCP_ESTATS_BANDWIDTH_RW_v0
TcpConnectionEstatsFineRtt	TCP_ESTATS_FINE_RTT_RW_v0

Offset 参数当前未使用，必须设置为 0。*Rw* 参数指向的可能结构都具有单个成员 ([TCP_ESTATS_BANDWIDTH_RW_v0](#) 结构除外)。当 *EstatsType* 参数设置为 [TcpConnectionEstatsBandwidth](#) 时，*Rw* 参数指向的 [TCP_ESTATS_BANDWIDTH_RW_v0](#) 结构必须在对 [SetPerTcpConnectionEStats](#) 函数的单个调用中将两个结构成员都设置为首选值。

如果 *RwSize* 参数设置为 0，则 [SetPerTcpConnectionEStats](#) 函数将返回 [NO_ERROR](#)，并且不会对扩展统计信息状态进行更改。

[GetTcpTable](#) 函数用于检索本地计算机上的 IPv4 TCP 连接表。此函数返回包含 [MIB_TCPROW](#) 项数组的 [MIB_TCPTABLE](#) 结构。传递给 [SetPerTcpConnectionEStats](#) 函数的 *Row* 参数必须是现有 IPv4 TCP 连接的条目。

在 IPv4 的 TCP 连接上启用扩展统计信息后，应用程序将调用 [GetPerTcpConnectionEStats](#) 函数来检索 TCP 连接上的扩展统计信息。

[GetPerTcpConnectionEStats](#) 函数旨在使用 TCP 诊断网络和应用程序中的性能问题。如果基于网络的应用程序性能不佳，TCP 可以确定瓶颈在发送方、接收方还是网络本身。如果瓶颈在网络中，TCP 可以提供有关其性质的特定信息。

有关 IPv6 连接的扩展 TCP 统计信息的信息，请参阅 [GetPerTcp6ConnectionEStats](#) 和 [SetPerTcp6ConnectionEStats](#) 函数。

[SetPerTcpConnectionEStats](#) 函数只能由以管理员组成员身份登录的用户调用。如果 [SetPerTcpConnectionEStats](#) 由不是 Administrators 组成员的用户调用，则函数调用将失

败并返回 `ERROR_ACCESS_DENIED`。此函数也可能因为 Windows Vista 和 Windows Server 2008 上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 `requestedExecutionLevel` 设置为 `requireAdministrator`。如果 Windows Vista 或 Windows Server 2008 上的应用程序缺少此清单文件，则作为内置管理员以外的管理员组成员登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

在 Windows 10 版本 1709 (Fall Creators Update) 之前的 Windows 版本中，可以使用 `SetPerTcpConnectionEStats` 禁用和重新启用连接上的统计信息，从而导致任何统计信息计数器重置为零。从 Windows 10 版本 1709 (Fall Creators Update) 起，某些统计信息计数器在禁用和重新启用时不会重置。此外，如果两个应用程序正在监视同一连接的统计信息，则一个应用程序可能会通过禁用统计信息来混淆另一个应用程序。出于这些原因，我们建议应用程序不要禁用有关连接的统计信息。若要检测一段时间内的更改，应保存上一次调用 `GetPerTcpConnectionEStats` 读取的计数器值，并从后续调用读取的计数器值中减去这些值。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	<code>iphlpapi.h</code>
Library	<code>iphlpapi.lib</code>
DLL	<code>iphlpapi.dll</code>

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[GetTcpTable](#)

[MIB_TCPROW](#)

[SetPerTcp6ConnectionEStats](#)

TCP_BOOLEAN_OPTIONAL

TCP_ESTATS_BANDWIDTH_RW_v0

TCP_ESTATS_DATA_RW_v0

TCP_ESTATS_FINE_RTT_RW_v0

TCP_ESTATS_OBS_REC_RW_v0

TCP_ESTATS_PATH_RW_v0

TCP_ESTATS_REC_RW_v0

TCP_ESTATS_SEND_BUFF_RW_v0

TCP_ESTATS SND CONG RW_v0

TCP_ESTATS_TYPE

反馈

此页面是否有帮助?



是



否

在 Microsoft Q&A 获得帮助

setTcpEntry 函数 (iphlpapi.h)

项目2023/08/24

SetTcpEntry 函数设置 TCP 连接的状态。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD SetTcpEntry(
    [in] PMIB_TCPROW pTcpRow
);
```

parameters

[in] pTcpRow

指向 [MIB_TCPROW](#) 结构的指针。此结构指定用于标识要修改的 TCP 连接的信息。它还指定 TCP 连接的新状态。调用方必须为此结构中的所有成员指定值。

返回值

如果函数成功，则函数返回 [NO_ERROR](#) (零)。

如果函数失败，则返回值为以下错误代码之一。

返回代码/值	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在 Windows Vista 和 Windows Server 2008 上返回，条件如下：用户在本地计算机上缺少所需的管理权限，或者应用程序未在增强的 shell 中运行，因为内置管理员 (RunAs 管理员)。
ERROR_INVALID_PARAMETER	输入参数无效，未执行任何操作。如果 <i>pTcpRow</i> 参数为 NULL，或者 <i>pTcpRow</i> 参数指向的 MIB_TCPROW 结构中的 Row 成员未设置为 MIB_TCP_STATE_DELETE_TCB ，则返回此错误。
ERROR_NOT_SUPPORTED	未在本地计算机上配置 IPv4 传输。
317	函数无法设置 TCP 条目，因为应用程序运行非提升。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

目前，TCP 连接可以设置为的唯一状态是 MIB_TCP_STATE_DELETE_TCB。

在 Windows Vista 及更高版本上，SetTcpEntry 函数只能由以 Administrators 组成员身份登录的用户调用。如果 SetTcpEntry 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。

SetTcpEntry 函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而不是内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而不是内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 此函数才能成功。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	iphlpapi.h
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

另请参阅

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[MIB_TCPROW](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

SetUnicastIpAddressEntry 函数 (netioapi.h)

项目2023/08/25

[SetUnicastIpAddressEntry](#) 函数设置本地计算机上现有单播 IP 地址条目的属性。

语法

C++

```
IPHLPAPI_DLL_LINKAGE _NETIOAPI_SUCCESS_ NETIOAPI_API
SetUnicastIpAddressEntry(
    [in] const MIB_UNICASTIPADDRESS_ROW *Row
);
```

parameters

[in] Row

指向现有单播 IP 地址条目 的[MIB_UNICASTIPADDRESS_ROW](#) 结构条目的指针。

返回值

如果函数成功，则返回值NO_ERROR。

如果函数失败，则返回值为以下错误代码之一。

返回代码	说明
ERROR_ACCESS_DENIED	拒绝访问。此错误在以下几种情况下返回：用户在本地计算机上缺少所需的管理权限，或者应用程序没有作为内置管理员 (RunAs 管理员) 在增强的 shell 中运行。
ERROR_INVALID_PARAMETER	向该函数传递了无效参数。如果在 Row 参数中传递 NULL 指针，Row 参数指向 MIB_UNICASTIPADDRESS_ROW 的 Address 成员未设置为有效的单播 IPv4 或 IPv6 地址，或者 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员均未指定，则返回此错误。 对于为 MIB_UNICASTIPADDRESS_ROW 结构中的成员设置的值中的其他错误，也会返回此错误。这些错误包括以下

	错误：如果 ValidLifetime 成员小于 PreferredLifetime 成员，如果 PrefixOrigin 成员设置为 IpPrefixOriginUnchanged，SuffixOrigin 未设置为 IpSuffixOriginUnchanged，如果 PrefixOrigin 成员未设置为 IpPrefixOriginUnchanged，并且 SuffixOrigin 设置为 IpSuffixOriginUnchanged，如果 PrefixOrigin 如果 SuffixOrigin 成员未设置为 NL_SUFFIX_ORIGIN 枚举中的值，或者如果 OnLinkPrefixLength 成员设置为大于 IP 地址长度的值，则不会将 member 设置为 NL_PREFIX_ORIGIN 枚举中的值，对于单播 IPv4 地址，以位 (32 或 128 表示单播 IPv6 地址)。
ERROR_NOT_FOUND	找不到指定的接口。如果找不到由 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW 的 InterfaceLuid 或 InterfaceIndex 成员指定的网络接口，则返回此错误。
ERROR_NOT_SUPPORTED	不支持该请求。如果本地计算机上没有 IPv4 堆栈，且在 Row 参数指向的 Address 成员 MIB_UNICASTIPADDRESS_ROW 指定了 IPv4 地址，或者本地计算机上没有 IPv6 堆栈，并且 地址 成员中指定了 IPv6 地址，则返回此错误。
其他	使用 FormatMessage 获取返回错误的消息字符串。

注解

[SetUnicastIpAddressEntry](#) 函数在 Windows Vista 及更高版本上定义。

[GetUnicastIpAddressEntry](#) 函数通常用于检索要修改的现有 [MIB_UNICASTIPADDRESS_ROW](#) 结构条目。然后，应用程序可以更改要修改的 [MIB_UNICASTIPADDRESS_ROW](#) 条目中的成员，然后调用 [SetUnicastIpAddressEntry](#) 函数。

应用程序可以调用 [InitializeUnicastIpAddressEntry](#) 函数，以在进行更改之前使用默认值初始化 [MIB_UNICASTIPADDRESS_ROW](#) 结构条目的成员。但是，应用程序通常会在调用 [InitializeUnicastIpAddressEntry](#) 之前保存 [InterfaceLuid](#) 或 [InterfaceIndex](#) 成员，并在调用后还原其中一个成员。

Row 参数指向的 [MIB_UNICASTIPADDRESS_ROW](#) 结构中的 Address 成员必须初始化为有效的单播 IPv4 或 IPv6 地址和系列。此外，必须初始化指向 Row 参数的 [MIB_UNICASTIPADDRESS_ROW](#) 结构中的以下至少一个成员：[InterfaceLuid](#) 或 [InterfaceIndex](#)。

字段按上面列出的顺序使用。因此，如果指定了 [InterfaceLuid](#)，则使用此成员来确定接口。如果没有为 [InterfaceLuid](#) 成员设置值，(此成员的值) 设置为零，则接下来使用 [InterfaceIndex](#) 成员来确定接口。

如果 Row 参数指向的 MIB_UNICASTIPADDRESS_ROW OnLinkPrefixLength 成员设置为 255，则 SetUnicastIpAddressEntry 将设置单播 IP 地址属性，以便 OnLinkPrefixLength 成员等于 IP 地址的长度。因此，对于单播 IPv4 地址，对于单播 IPv6 地址，OnLinkPrefixLength 设置为 32，OnLinkPrefixLength 设置为 128。如果这会导致 IPv4 地址的子网掩码不正确或 IPv6 地址的链接前缀不正确，则在调用 SetUnicastIpAddressEntry 之前，应用程序应将此成员设置为正确的值。

调用 SetUnicastIpAddressEntry 函数时，将忽略行指向的 MIB_UNICASTIPADDRESS_ROW 结构的 DadState、ScopId 和 CreationTimeStamp 成员。这些成员由网络堆栈设置，不能使用 SetUnicastIpAddressEntry 函数进行更改。ScopId 成员由添加地址的接口自动确定。

SetUnicastIpAddressEntry 函数只能由以管理员组成员身份登录的用户调用。如果 SetUnicastIpAddressEntry 由不是 Administrators 组成员的用户调用，则函数调用将失败并返回 ERROR_ACCESS_DENIED。

SetUnicastIpAddressEntry 函数也可能因为 Windows Vista 及更高版本上的用户帐户控制 (UAC) 而失败。如果包含此函数的应用程序由以管理员组成员身份登录（而非内置管理员）的用户执行，则此调用将失败，除非应用程序已在清单文件中标记为 requestedExecutionLevel 设置为 requireAdministrator。如果应用程序缺少此清单文件，则作为管理员组成员（而非内置管理员）登录的用户必须在增强的 shell 中执行应用程序，因为内置管理员 (RunAs 管理员) 才能使此功能成功。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
目标平台	Windows
标头	netioapi.h (包括 Iphlpapi.h)
Library	Iphlpapi.lib
DLL	Iphlpapi.dll

请参阅

[CreateUnicastIpAddressEntry](#)

[DeleteUnicastIpAddressEntry](#)

[GetUnicastIpAddressEntry](#)

[GetUnicastIpAddressTable](#)

IP 帮助程序函数参考

[InitializeUnicastIpAddressEntry](#)

[MIB_UNICASTIPADDRESS_ROW](#)

[MIB_UNICASTIPADDRESS_TABLE](#)

[NotifyUnicastIpAddressChange](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

UnenableRouter 函数 (iphlpapi.h)

项目2023/08/24

`UnenableRouter` 函数递减用于跟踪启用 IPv4 转发的请求数的引用计数。 当此引用计数达到零时，`UnenableRouter` 会关闭本地计算机上的 IPv4 转发。

语法

C++

```
IPHLPAPI_DLL_LINKAGE DWORD UnenableRouter(
    OVERLAPPED *pOverlapped,
    [out, optional] LPDWORD lpdwEnableCount
);
```

parameters

pOverlapped

指向 `OVERLAPPED` 结构的指针。 此结构应与调用 [EnableRouter](#) 函数中使用的结构相同。

[out, optional] lpdwEnableCount

指向 `DWORD` 变量的可选指针。 此变量接收剩余引用数。

返回值

如果函数成功，则返回值 `NO_ERROR`。

如果函数失败，请使用 [FormatMessage](#) 获取返回错误的消息字符串。

注解

`UnenableRouter` 函数特定于 IPv4 转发。 进程对 `UnenableRouter` 进行的每个调用都必须对应于同一进程对 [EnableRouter](#) 的先前调用。 系统对 `UnenableRouter` 进行无关调用时返回错误。 因此，给定进程无法减少用于跟踪为另一个进程启用 IPv4 转发的请求数的引用计数。 此外，如果给定进程启用了 IPv4 转发，则其他进程无法禁用它。

无法准确确定用于跟踪启用 IPv4 转发的请求数的引用计数，因为可能存在其他未完成 的 [EnableRouter](#) 请求。因此，为 *lpdwEnableCount* 参数返回的值始终是等于 `ULONG_MAX/2` 的大计数。

如果调用 [EnableRouter](#) 的进程在未调用 [UnenableRouter](#) 的情况下终止，系统会减少跟踪启用 IPv4 转发的请求的引用计数，就像进程调用了 [UnenableRouter](#) 一样。

调用 [UnenableRouter](#) 后，使用 [CloseHandle](#) 调用关闭 [OVERLAPPED](#) 结构中事件对象的句柄。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
目标平台	Windows
标头	<code>iphlpapi.h</code>
Library	<code>Iphlpapi.lib</code>
DLL	<code>Iphlpapi.dll</code>

另请参阅

[CloseHandle](#)

[EnableRouter](#)

[IP 帮助程序函数参考](#)

[IP 帮助程序起始页](#)

[OVERLAPPED](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

UnregisterInterfaceTimestampConfigChange 函数 (iphlpapi.h)

项目2023/08/24

通过取消注册在调用 [RegisterInterfaceTimestampConfigChange](#) 中注册的回调函数，取消有关时间戳功能更改的通知。

若要避免死锁，不应在正在执行 [RegisterInterfaceTimestampConfigChange](#) 指定回调的线程上下文中调用 [UnregisterInterfaceTimestampConfigChange](#)。这是因为 [UnregisterInterfaceTimestampConfigChange](#) 等待未完成的通知回调，然后再返回。因此，不得以阻止未完成通知回调的方式调用 [UnregisterInterfaceTimestampConfigChange](#)。
[UnregisterInterfaceTimestampConfigChange](#) 返回后，不会再进行回调。

有关详细信息和代码示例，请参阅[数据包时间戳](#)。

语法

C++

```
IPHLPAPI_DLL_LINKAGE VOID UnregisterInterfaceTimestampConfigChange(
    HIFTIMESTAMPCHANGE NotificationHandle
);
```

parameters

NotificationHandle

类型: `_In_ HIFTIMESTAMPCHANGE`

[RegisterInterfaceTimestampConfigChange](#) 返回的句柄。这标识要取消的注册。

返回值

类型: `DWORD`

指示成功或失败的 `DWORD` 返回代码。

要求

最低受支持的客户端	Windows 10内部版本 20348
最低受支持的服务器	Windows 10内部版本 20348
目标平台	Windows
标头	iphlpapi.h
Library	iphlpapi.lib
DLL	iphlpapi.dll

另请参阅

- [包时间戳](#)
- [RegisterInterfaceTimestampConfigChange](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP Helper 结构

项目 • 2023/06/13

以下结构和联合与 IP 帮助程序一起使用。

- ARP_SEND_REPLY
- DNS_DOH_SERVER_SETTINGS
- DNS_INTERFACE_SETTINGS
- DNS_INTERFACE_SETTINGS3
- DNS_SERVER_PROPERTY_TYPES
- DNS_SERVER_PROPERTY
- FIXED_INFO
- ICMP_ECHO_REPLY
- ICMP_ECHO_REPLY32
- ICMP_ERROR_INFO
- ICMPV6_ECHO_REPLY
- in_addr
- INTERFACE_HARDWARE_TIMESTAMP_CAPABILITIES
- INTERFACE_HARDWARE_CROSSTIMESTAMP
- INTERFACE_SOFTWARE_TIMESTAMP_CAPABILITIES
- INTERFACE_TIMESTAMP_CAPABILITIES
- IP_ADAPTER_ADDRESSES
- IP_ADAPTER_ANYCAST_ADDRESS
- IP_ADAPTER_DNS_SERVER_ADDRESS
- IP_ADAPTER_DNS_SUFFIX
- IP_ADAPTER_GATEWAY_ADDRESS
- IP_ADAPTER_INDEX_MAP
- IP_ADAPTER_INFO
- IP_ADAPTER_MULTICAST_ADDRESS
- IP_ADAPTER_ORDER_MAP
- IP_ADAPTER_PREFIX
- IP_ADAPTER_UNICAST_ADDRESS
- IP_ADAPTER_WINS_SERVER_ADDRESS
- IP_ADDR_STRING
- IP_ADDRESS_PREFIX
- IP_ADDRESS_STRING
- IP_INTERFACE_INFO
- IP_INTERFACE_NAME_INFO
- IP_MCAST_COUNTER_INFO
- IP_OPTION_INFORMATION

- IP_OPTION_INFORMATION32
- IP_PER_ADAPTER_INFO
- IP_UNIDIRECTIONAL_ADAPTER_ADDRESS
- IPV6_ADDRESS_EX
- NET_ADDRESS_INFO
- NET_LUID
- NL_NETWORK_CONNECTIVITY_HINT
- SOCKADDR_IN6_PAIR
- SOCKADDR_INET
- TCP_ESTATS_BANDWIDTH_ROD_v0
- TCP_ESTATS_BANDWIDTH_RW_v0
- TCP_ESTATS_DATA_ROD_v0
- TCP_ESTATS_DATA_RW_v0
- TCP_ESTATS_FINE_RTT_ROD_v0
- TCP_ESTATS_FINE_RTT_RW_v0
- TCP_ESTATS_OBS_REC_ROD_v0
- TCP_ESTATS_OBS_REC_RW_v0
- TCP_ESTATS_PATH_ROD_v0
- TCP_ESTATS_PATH_RW_v0
- TCP_ESTATS_REC_ROD_v0
- TCP_ESTATS_REC_RW_v0
- TCP_ESTATS_SEND_BUFF_ROD_v0
- TCP_ESTATS_SEND_BUFF_RW_v0
- TCP_ESTATS SND CONG ROD v0
- TCP_ESTATS SND CONG ROS v0
- TCP_ESTATS SND CONG RW v0
- TCP_ESTATS_SYN_OPTS_ROS_v0
- TCPIP_OWNER_MODULE_BASIC_INFO
- TCP_RESERVE_PORT_RANGE

反馈

此页面是否有帮助?

是

否

在 Microsoft Q&A 获得帮助

ARP_SEND_REPLY 结构 (ipexport.h)

项目2023/08/24

ARP_SEND_REPLY结构存储有关地址解析协议的信息 (ARP) 答复消息。

语法

C++

```
typedef struct arp_send_reply {
    IPAddr DestAddress;
    IPAddr SrcAddress;
} ARP_SEND_REPLY, *PARP_SEND_REPLY;
```

成员

DestAddress

ARP 消息发送到的目标 IPv4 地址，采用 [IPAddr](#) 结构的形式。

SrcAddress

从中传输 ARP 消息的源 IPv4 地址，采用 [IPAddr](#) 结构的形式。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IPAddr](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

netioapi.h) (DNS_DOH_SERVER_SETTINGS 结构

项目2023/08/25

描述 DNS-over-HTTPS 服务器。

语法

C++

```
typedef struct _DNS_DOH_SERVER_SETTINGS {
#if ...
    PWSTR    Template;
#else
    PWSTR    Template;
#endif
    ULONG64 Flags;
} DNS_DOH_SERVER_SETTINGS;
```

成员

Template

类型: [PWSTR](#)

以 NULL 结尾的宽字符串，其中包含有效的 DNS-over-HTTPS URI 模板。

如果 **存在**DNS_DOH_SERVER_SETTINGS_ENABLE_AUTO 标志，则此字段必须为 NULL。

① 重要

URI 模板不得包含与所引用服务器的 IP 地址不同的 IP 地址作为主机名。例如，如果引用的服务器为 1.1.1.1，URI 模板为 `https://1.0.0.1/dns-query`，则无效，因为 1.0.0.1 与服务器 IP 1.1.1.1 不匹配。

Flags

类型: [ULONG64](#)

包含以下任何选项的位图。

DNS_DOH_SERVER_SETTINGS_ENABLE_AUTO (0x0001)。如果存在此选项，则此属性引用的 DNS 服务器将从系统 DNS-over-HTTPS 系统列表中加载其 URI 模板。如果存在此选项，则必须将“模板”字段设置为 NULL。此选项不得与 **DNS_DOH_SERVER_SETTINGS_ENABLE** 选项一起使用。

DNS_DOH_SERVER_SETTINGS_ENABLE (0x0002)。如果存在此选项，则“模板”字段必须指向有效的 DNS-over-HTTPS URI 模板。此选项不得与 **DNS_DOH_SERVER_SETTINGS_ENABLE_AUTO** 选项一起使用。

DNS_DOH_SERVER_SETTINGS_FALLBACK_TO_UDP (0x0004)。此选项指示如果 DNS-over-HTTPS 查询失败，引用的服务器可能会回退到不安全的名称解析 (UDP/TCP)。除了 **DNS_DOH_SERVER_SETTINGS_ENABLE_AUTO** 或 **DNS_DOH_SERVER_SETTINGS_ENABLE** 之外，此选项只能使用。

DNS_DOH_AUTO_UPGRADE_SERVER (0x0008)。如果 NRPT 规则中存在的 DNS 服务器与此属性引用的服务器具有相同的 IP 地址，则此选项允许该服务器使用 DNS-over-HTTPS 模板。此选项不能单独使用；它必须是 **DNS_DOH_SERVER_SETTINGS_ENABLE_AUTO** 或 **DNS_DOH_SERVER_SETTINGS_ENABLE** 的补充。

要求

最低受支持的客户端	无受支持的版本
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	netioapi.h (包括 Iphlpapi.h)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

netioapi.h) (DNS_INTERFACE_SETTINGS 结构)

项目2023/04/27

表示可通过调用 [SetInterfaceDnsSettings](#) 函数在给定接口上配置的 DNS 设置，或通过调用 [GetInterfaceDnsSettings](#) 函数为给定接口检索这些设置。

语法

C++

```
typedef struct _DNS_INTERFACE_SETTINGS {
    ULONG    Version;
    ULONG64  Flags;
    PWSTR    Domain;
    PWSTR    NameServer;
    PWSTR    SearchList;
    ULONG    RegistrationEnabled;
    ULONG    RegisterAdapterName;
    ULONG    EnableLLMNR;
    ULONG    QueryAdapterName;
    PWSTR    ProfileNameServer;
} DNS_INTERFACE_SETTINGS;
```

成员

Version

类型: [ULONG](#)

必须设置为 [DNS_INTERFACE_SETTINGS_VERSION1](#)。

Flags

类型: [ULONG64](#)

以下选项的位图。

[DNS_SETTING_IPV6](#) (0x0001)。仅为 IPv6 网络堆栈配置接口设置。如果设置了此选项，则 *NameServer* 或 *ProfileNameServer* 成员中指定的任何 IP 地址必须是 IPv6 地址。默认情况下，在此结构中指定的 DNS 接口设置仅应用于 IPv4 网络堆栈。

DNS_SETTING_NAMESERVER (0x0002)。通过 *NameServer* 成员在指定接口上配置静态适配器 DNS 服务器。

DNS_SETTING_SEARCHLIST (0x0004)。通过 *SearchList* 成员为给定适配器配置特定于连接的 DNS 后缀搜索列表。

DNS_SETTING_REGISTRATION_ENABLED (0x0008)。启用或禁用给定适配器的动态 DNS 注册。默认情况下，这是系统启用的。

DNS_SETTING_DOMAIN (0x0020)。通过 *域* 成员为给定适配器配置特定于连接的 DNS 后缀。

DNS_SETTINGS_ENABLE_LLMNR (0x0080)。在指定的适配器上使用 LLMNR 和 mDNS 启用或禁用名称解析。默认情况下，这是系统启用的。

DNS_SETTINGS_QUERY_ADAPTER_NAME (0x0100)。启用或禁用将适配器名称用作 DNS 查询的后缀。默认情况下，这是系统启用的。

DNS_SETTING_PROFILE_NAMESERVER (0x0200)。通过 *ProfileNameServer* 成员在指定接口上配置静态配置文件 DNS 服务器。

Domain

类型: [PWSTR](#)

包含适配器域名的以 NULL 结尾的宽字符串。

NameServer

类型: [PWSTR](#)

以 NULL 结尾的宽字符串，其中包含一系列逗号或空格分隔的 DNS 服务器。例如，L"1.1.1.1 8.8.8" 或 L"1.1.1.1, 8.8.8.8"。

如果 存在**DNS_SETTING_IPV6** 标志，则服务器必须是 IPv6 地址。例如，L"2606:4700:4700::1001, 2606:4700:4700::1111"。

SearchList

类型: [PWSTR](#)

以 NULL 结尾的宽字符串，其中包含一系列逗号或空格分隔的搜索名称。例如，L"contoso1.com contoso2.com" 或 L"contoso1.com, contoso2.com"。

RegistrationEnabled

类型: [ULONG](#)

如果为 TRUE，则启用适配器动态注册；**如果为 FALSE**，则禁用它。

RegisterAdapterName

类型： **ULONG**

如果为 TRUE，则启用适配器名称注册；**如果为 FALSE**，则禁用它。

EnableLLMNR

类型： **ULONG**

若要 在给定接口上启用 mDNS 和 LLMNR，则为 **TRUE**；**如果为 FALSE**，则禁用它们。

QueryAdapterName

类型： **ULONG**

如果适配器名称应用作搜索后缀，则为 **TRUE**；否则为 **FALSE**。

ProfileNameServer

类型： **PWSTR**

以 NULL 结尾的宽字符串，其中包含一系列逗号或空格分隔的 DNS 服务器。例如，L“1.1.1.1 8.8.8”或 L“1.1.1.1, 8.8.8.8”。

如果 **存在**DNS_SETTING_IPV6 标志，则服务器必须是 IPv6 地址。例如，L“2606:4700:4700::1001, 2606:4700:4700::1111”。

要求

最低受支持的客户端	Windows 10内部版本 18362
最低受支持的服务器	Windows 10内部版本 18362
标头	netioapi.h (包括 Iphlpapi.h)

另请参阅

- [GetInterfaceDnsSettings](#)
- [SetInterfaceDnsSettings](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

DNS_INTERFACE_SETTINGS3 结构 (netioapi.h)

项目2023/04/27

表示可以通过调用 [SetInterfaceDnsSettings](#) 函数在给定接口上配置的 DNS 设置，或通过调用 [GetInterfaceDnsSettings](#) 函数为给定接口检索的 DNS 设置。

语法

C++

```
typedef struct _DNS_INTERFACE_SETTINGS3 {
    ULONG             Version;
    ULONG64           Flags;
    PWSTR             Domain;
    PWSTR             NameServer;
    PWSTR             SearchList;
    ULONG             RegistrationEnabled;
    ULONG             RegisterAdapterName;
    ULONG             EnableLLMNR;
    ULONG             QueryAdapterName;
    PWSTR             ProfileNameServer;
    ULONG             DisableUnconstrainedQueries;
    PWSTR             SupplementalSearchList;
    ULONG             cServerProperties;
    DNS_SERVER_PROPERTY *ServerProperties;
    ULONG             cProfileServerProperties;
    DNS_SERVER_PROPERTY *ProfileServerProperties;
} DNS_INTERFACE_SETTINGS3;
```

成员

Version

类型: [ULONG](#)

必须设置为 [DNS_INTERFACE_SETTINGS_VERSION3](#)。

Flags

类型: [ULONG64](#)

以下选项的位图。

DNS_SETTING_IPV6 (0x0001)。仅为 IPv6 网络堆栈配置接口设置。如果设置了此选项，则 *NameServer* 或 *ProfileNameServer* 成员中指定的任何 IP 地址都必须是 IPv6 地址。默认情况下，此结构中指定的 DNS 接口设置仅应用于 IPv4 网络堆栈。

DNS_SETTING_NAMESERVER (0x0002)。通过 *NameServer* 成员在指定接口上配置静态适配器 DNS 服务器。

DNS_SETTING_SEARCHLIST (0x0004)。通过 *SearchList* 成员为给定适配器配置特定于连接的 DNS 后缀搜索列表。

DNS_SETTING_REGISTRATION_ENABLED (0x0008)。启用或禁用给定适配器的动态 DNS 注册。默认情况下，这是系统启用的。

DNS_SETTING_DOMAIN (0x0020)。通过 *域* 成员为给定适配器配置特定于连接的 DNS 后缀。

DNS_SETTINGS_ENABLE_LLMNR (0x0080)。在指定的适配器上使用 LLMNR 和 mDNS 启用或禁用名称解析。默认情况下，这是系统启用的。

DNS_SETTINGS_QUERY_ADAPTER_NAME (0x0100)。启用或禁用将适配器名称用作 DNS 查询的后缀。默认情况下，这是系统启用的。

DNS_SETTING_PROFILE_NAMESERVER (0x0200)。通过 *ProfileNameServer* 成员在指定接口上配置静态配置文件 DNS 服务器。

DNS_SETTING_SUPPLEMENTAL_SEARCH_LIST (0x0800)。通过 *SupplementalSearchList* 成员为给定适配器配置特定于连接的 DNS 补充后缀搜索列表。

DNS_SETTING_DOH (0x1000)。通过 *cServerProperties* 和 *ServerProperties* 成员在指定适配器上配置 DNS-over-HTTPS 设置。如果设置了此选项，则 *NameServer* 成员必须指向包含一系列空格或逗号分隔的 DNS 服务器的有效字符串。

DNS_SETTING_DOH_PROFILE (0x2000)。通过 *cProfileServerProperties* 和 *ProfileServerProperties* 成员在指定适配器上配置 配置文件 DNS-over-HTTPS 设置。如果设置了此选项，则 *ProfileNameServer* 成员必须指向包含一系列空格或逗号分隔的 DNS 服务器的有效字符串。

Domain

类型： **PWSTR**

包含适配器域名的以 NULL 结尾的宽字符串。

NameServer

类型： **PWSTR**

以 NULL 结尾的宽字符串，包含一系列逗号或空格分隔的 DNS 服务器。例如，L"1.1.1.1 8.8.8"或 L"1.1.1, 8.8.8"。

如果 存在DNS_SETTING_IPV6 标志，则服务器必须是 IPv6 地址。例如，L"2606: 4700: 4700: : 1001, 2606: 4700: 4700: : 1111"。

SearchList

类型: **PWSTR**

以 NULL 结尾的宽字符串，包含一系列逗号或空格分隔的搜索名称。例如，L"contoso1.com contoso2.com"或 L"contoso1.com, contoso2.com"。

RegistrationEnabled

类型: **ULONG**

如果为 TRUE，则启用适配器动态注册; 如果为 FALSE，则禁用它。

RegisterAdapterName

类型: **ULONG**

如果为 TRUE，则启用适配器名称注册; 如果为 FALSE，则禁用它。

EnableLLMNR

类型: **ULONG**

如果为 TRUE，则为在给定接口上启用 mDNS 和 LLMNR; 如果为 FALSE，则禁用它们。

QueryAdapterName

类型: **ULONG**

如果适配器名称应用作搜索后缀，则为 TRUE;否则为 FALSE。

ProfileNameServer

类型: **PWSTR**

以 NULL 结尾的宽字符串，包含一系列逗号或空格分隔的 DNS 服务器。例如，L"1.1.1.1 8.8.8"或 L"1.1.1, 8.8.8"。

如果 存在DNS_SETTING_IPV6 标志，则服务器必须是 IPv6 地址。例如，L"2606: 4700: 4700: : 1001, 2606: 4700: 4700: : 1111"。

DisableUnconstrainedQueries

类型: **ULONG**

保留。

`SupplementalSearchList`

类型: **PWSTR**

以 NULL 结尾的宽字符串, 包含一系列逗号或空格分隔的搜索名称。例如, L"contoso1.com contoso2.com" 或 L"contoso1.com, contoso2.com"。

`cServerProperties`

类型: **ULONG**

ServerProperties 成员中指定的服务器属性数。如果此值等于 0, 则“*ServerProperties*”成员必须为 NULL。

`ServerProperties`

类型: **DNS_SERVER_PROPERTY***

包含 *cServerProperties* 元素的 DNS_SERVER_PROPERTY 结构的数组。如果 *cServerProperties* 为 0, 则此值必须为 NULL。

仅支持 DNS-over-HTTPS 属性, 在 *NameServer* 成员中指定的每个服务器最多只能有 1 个属性。

DNS_SERVER_PROPERTY: : *Version* 成员必须设置为

DNS_SERVER_PROPERTY_VERSION1, : *Type* 必须设置为

DNS_SERVER_PROPERTY_TYPE: :D *nsServerDohProperty*, *Property.DohSettings* 必须指向有效的 **DNS_DOH_SERVER_SETTINGS** 对象。

必须将 DNS_SERVER_PROPERTY 的 *ServerIndex* 成员设置为 *NameServer* 成员中相应 DNS 服务器的索引。

例如, 如果 *NameServer* 成员设置为 L"1.1.1.1、8.8.8、9.9.9.9", 则服务器 1.1.1.1 的属性会将 *ServerIndex* 成员设置为 0。同样, 8.8.8.8 需要 将 *ServerIndex* 设置为 1, 而 9.9.9.9 需要 将 *ServerIndex* 成员设置为 2。

`cProfileServerProperties`

类型: **ULONG**

在 *ProfileServerProperties* 成员中指定的服务器属性数。如果此值等于 0, 则 *ProfileServerProperties* 成员必须为 NULL。

ProfileServerProperties

类型: [DNS_SERVER_PROPERTY*](#)

包含 *cProfileServerProperties* 元素的 DNS_SERVER_PROPERTY 结构的数组。如果 *cProfileServerProperties* 为 0，则此值必须为 NULL。

仅支持 DNS-over-HTTPS 属性，在 *ProfileNameServer* 成员中指定的每个服务器最多只能有 1 个属性。

DNS_SERVER_PROPERTY: : Version 成员必须设置为

DNS_SERVER_PROPERTY_VERSION1, : Type 必须设置为

DNS_SERVER_PROPERTY_TYPE: :D nsServerDohProperty, Property.DohSettings 必须指向有效的 [DNS_DOH_SERVER_SETTINGS](#) 对象。

必须将 DNS_SERVER_PROPERTY 的 ServerIndex 成员设置为 *ProfileNameServer* 成员中相应 DNS 服务器的索引。

例如，如果 *ProfileNameServer* 成员设置为 L“1.1.1.1、8.8.8、9.9.9.9”，则服务器 1.1.1.1 的属性会将 *ServerIndex* 成员设置为 0。同样，8.8.8.8 需要 将 *ServerIndex* 设置为 1，而 9.9.9.9 需要 将 *ServerIndex* 成员设置为 2。

要求

最低受支持的客户端	Windows 10 内部版本 19645
最低受支持的服务器	Windows 10 内部版本 19645
标头	netioapi.h (包括 Iphlpapi.h)

另请参阅

- [GetInterfaceDnsSettings](#)
- [SetInterfaceDnsSettings](#)

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获取帮助

DNS_SERVER_PROPERTY_TYPES union (netioapi.h)

项目2023/03/16

包含指向 DNS 服务器属性的指针。 属性的类型取决于 [DNS_SERVER_PROPERTY](#) : : Type 的值。

语法

C++

```
typedef union _DNS_SERVER_PROPERTY_TYPES {
    DNS_DOH_SERVER_SETTINGS *DohSettings;
} DNS_SERVER_PROPERTY_TYPES;
```

成员

DohSettings

如果 [DNS_SERVER_PROPERTY](#) : : Type 设置为 *DnsServerDohProperty*，则 *DohSettings* 将指向有效的 DNS-over-HTTPS 服务器属性。

要求

最低受支持的客户端	无受支持的版本
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	netioapi.h (包括 Iphlpapi.h)

另请参阅

- [DNS_SERVER_PROPERTY](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

DNS_SERVER_PROPERTY 结构 (netioapi.h)

项目2023/03/09

描述 DNS 服务器属性，该属性在 [DNS_INTERFACE_SETTINGS3](#) 结构中设置，并通过 [SetInterfaceDnsSettings](#) 函数进行配置。

语法

C++

```
typedef struct _DNS_SERVER_PROPERTY {
    ULONG             Version;
    ULONG             ServerIndex;
    DNS_SERVER_PROPERTY_TYPE Type;
#if ...
    DNS_SERVER_PROPERTY_TYPES Property;
#else
    DNS_SERVER_PROPERTY_TYPES Property;
#endif
} DNS_SERVER_PROPERTY;
```

成员

Version

类型: [ULONG](#)

必须设置为 [DNS_INTERFACE_SETTINGS_VERSION1](#)。

ServerIndex

类型: [ULONG](#)

必须是 [DNS_INTERFACE_SETTINGS3](#) : : NameServer 或 : : ProfileNameServer 成员中存在的相应服务器的索引。有关正确用法，请参阅主题中的 *ServerProperties* 和 *ProfileServerProperties* 成员，[了解DNS_INTERFACE_SETTINGS3](#) 结构。

Type

类型: [DNS_SERVER_PROPERTY_TYPE](#)

必须设置为 [DnsServerDohProperty](#)。描述 DNS-over-HTTPS 服务器属性。

Property

类型: [DNS_SERVER_PROPERTY_TYPES](#)

如果 *Type* 成员设置为 `DnsServerDohProperty`, 则 `DNS_SERVER_PROPERTY_TYPES::DoHSettings` 字段必须指向有效的 `DNS_DOH_SERVER_SETTINGS` 对象。

要求

最低受支持的客户端	无受支持的版本
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	<code>netioapi.h</code> (包括 <code>Iphlpapi.h</code>)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

FIXED_INFO_W2KSP1结构 (iptypes.h)

项目2023/08/24

FIXED_INFO 结构包含计算机上所有接口中相同的信息。

语法

C++

```
typedef struct {
    char          HostName[MAX_HOSTNAME_LEN + 4];
    char          DomainName[MAX_DOMAIN_NAME_LEN + 4];
    PIP_ADDR_STRING CurrentDnsServer;
    IP_ADDR_STRING DnsServerList;
    UINT          NodeType;
    char          ScopeId[MAX_SCOPE_ID_LEN + 4];
    UINT          EnableRouting;
    UINT          EnableProxy;
    UINT          EnableDns;
} FIXED_INFO_W2KSP1, *PFIXED_INFO_W2KSP1;
```

成员

HostName[MAX_HOSTNAME_LEN + 4]

类型: `char[MAX_HOSTNAME_LEN + 4]`

本地计算机的主机名。 这可能是完全限定的主机名 (包括已加入域的计算机的域)。

DomainName[MAX_DOMAIN_NAME_LEN + 4]

类型: `char[MAX_DOMAIN_NAME_LEN + 4]`

在其中注册本地计算机的域。

CurrentDnsServer

类型: `PIP_ADDR_STRING`

保留。 使用 `DnsServerList` 成员获取本地计算机的 DNS 服务器。

DnsServerList

类型: `IP_ADDR_STRING`

[IP_ADDR_STRING](#)结构的链接列表，这些结构指定本地计算机使用的 DNS 服务器集。

NodeType

类型： **UINT**

本地计算机的节点类型。 这些值在 *Iptypes.h* 头文件中定义。

NodeType	含义
BROADCAST_NODETYPE 0x0001	广播节点类型。
PEER_TO_PEER_NODETYPE 0x0002	对等节点类型。
MIXED_NODETYPE 0x0004	混合节点类型。
HYBRID_NODETYPE 0x0008	混合节点类型。

ScopeId[MAX_SCOPE_ID_LEN + 4]

类型： **char[MAX_SCOPE_ID_LEN + 4]**

DHCP 作用域名称。

EnableRouting

类型： **UINT**

一个布尔值，指定是否在本地计算机上启用路由。

EnableProxy

类型： **UINT**

一个布尔值，指定本地计算机是否充当 ARP 代理。

EnableDns

类型： **UINT**

一个布尔值，指定是否在本地计算机上启用 DNS。

注解

FIXED_INFO 结构由 [GetNetworkParams](#) 函数检索。

在 Microsoft Windows 软件开发工具包 (SDK) 中，定义了 **FIXED_INFO_WIN2KSP1** 结构。如果目标平台是 Windows 2000 且 Service Pack 1 (SP1) 及更高版本 (`NTDDI_VERSION >= NTDDI_WIN2KSP1`、`_WIN32_WINNT >= 0x0501` 或 `WINVER >= 0x0501`)，则编译应用程序时，**FIXED_INFO_WIN2KSP1** 结构的类型为 **FIXED_INFO** 结构。如果目标平台不是 Windows 2000 SP1 及更高版本，则编译应用程序时，**FIXED_INFO** 结构未定义。

Windows 98 及更高版本支持 [GetNetworkParams](#) 函数和 **FIXED_INFO** 结构。但是，若要为早于 Windows 2000 且 Service Pack 1 (SP1) 的目标平台生成应用程序，必须使用早期版本的平台软件开发工具包 (SDK)。

示例

以下代码检索包含本地计算机的网络配置信息的 **FIXED_INFO** 结构。代码从结构中打印所选成员。

C++

```
//  
// Link with IPHlpAPI.lib  
//  
#include <winsock2.h>  
#include <iphlpapi.h>  
#include <stdio.h>  
#include <windows.h>  
#pragma comment(lib, "IPHLPAPI.lib")  
  
#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))  
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))  
  
/* Note: could also use malloc() and free() */  
  
int __cdecl main()  
{  
  
    FIXED_INFO *pFixedInfo;  
    ULONG ulOutBufLen;  
    DWORD dwRetVal;  
    IP_ADDR_STRING *pIPAddr;  
  
    pFixedInfo = (FIXED_INFO *) MALLOC(sizeof(FIXED_INFO));  
    if (pFixedInfo == NULL) {  
        printf("Error allocating memory needed to call GetNetworkParams\n");  
        return 1;  
    }  
    ulOutBufLen = sizeof(FIXED_INFO);  
  
    // Make an initial call to GetAdaptersInfo to get
```

```

// the necessary size into the ulOutBufLen variable
    if (GetNetworkParams(pFixedInfo, &ulOutBufLen) == ERROR_BUFFER_OVERFLOW)
{
    FREE(pFixedInfo);
    pFixedInfo = (FIXED_INFO *) MALLOC(ulOutBufLen);
    if (pFixedInfo == NULL) {
        printf("Error allocating memory needed to call
GetNetworkParams\n");
        return 1;
    }
}

if (dwRetVal = GetNetworkParams(pFixedInfo, &ulOutBufLen) == NO_ERROR) {

    printf("Host Name: %s\n", pFixedInfo->HostName);
    printf("Domain Name: %s\n", pFixedInfo->DomainName);

    printf("DNS Servers:\n");
    printf("\t%s\n", pFixedInfo->DnsServerList.IpAddress.String);

    pIPAddr = pFixedInfo->DnsServerList.Next;
    while (pIPAddr) {
        printf("\t%s\n", pIPAddr->IpAddress.String);
        pIPAddr = pIPAddr->Next;
    }

    printf("Node Type: ");
    switch (pFixedInfo->NodeType) {
    case BROADCAST_NODETYPE:
        printf("Broadcast node\n");
        break;
    case PEER_TO_PEER_NODETYPE:
        printf("Peer to Peer node\n");
        break;
    case MIXED_NODETYPE:
        printf("Mixed node\n");
        break;
    case HYBRID_NODETYPE:
        printf("Hybrid node\n");
        break;
    default:
        printf("Unknown node type %0lx\n", pFixedInfo->NodeType);
        break;
    }

    printf("DHCP scope name: %s\n", pFixedInfo->ScopeId);

    if (pFixedInfo->EnableRouting)
        printf("Routing: enabled\n");
    else
        printf("Routing: disabled\n");

    if (pFixedInfo->EnableProxy)
        printf("ARP proxy: enabled\n");
    else

```

```
    printf("ARP Proxy: disabled\n");

    if (pFixedInfo->EnableDns)
        printf("DNS: enabled\n");
    else
        printf("DNS: disabled\n");

} else {
    printf("GetNetworkParams failed with error: %d\n", dwRetVal);
    return 1;
}

if (pFixedInfo)
    FREE(pFixedInfo);

return 0;
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetNetworkParams](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADDR_STRING](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获取帮助](#)

ICMP_ECHO_REPLY结构 (ipexport.h)

项目2023/08/24

ICMP_ECHO_REPLY结构描述为响应 IPv4 回显请求而返回的数据。

语法

C++

```
typedef struct icmp_echo_reply {
    IPAddr Address;
    ULONG Status;
    ULONG RoundTripTime;
    USHORT DataSize;
    USHORT Reserved;
    PVOID Data;
    struct ip_option_information Options;
} ICMP_ECHO_REPLY, *PICMP_ECHO_REPLY;
```

成员

Address

类型: IPAddr

回复 IPv4 地址，采用 IPAddr 结构的形式。

Status

类型: ULONG

回显请求的状态，采用 IP_STATUS 代码的形式。此成员的可能值在 *Ipexport.h* 头文件中定义。

值	含义
IP_SUCCESS 0	状态为成功。
IP_BUF_TOO_SMALL 11001	回复缓冲区太小。
IP_DEST_NET_UNREACHABLE 11002	无法访问目标网络。

IP_DEST_HOST_UNREACHABLE 11003	无法访问目标主机。
IP_DEST_PROT_UNREACHABLE 11004	无法访问目标协议。
IP_DEST_PORT_UNREACHABLE 11005	无法访问目标端口。
IP_NO_RESOURCES 11006	可用 IP 资源不足。
IP_BAD_OPTION 11007	指定了错误的 IP 选项。
IP_HW_ERROR 11008	发生了硬件错误。
IP_PACKET_TOO_BIG 11009	数据包太大。
IP_REQ_TIMED_OUT 11010	请求超时。
IP_BAD_REQ 11011	错误的请求。
IP_BAD_ROUTE 11012	一条糟糕的路线。
IP_TTL_EXPIRED_TRANSIT 11013	(TTL 的生存时间) 在传输中过期。
IP_TTL_EXPIRED_REASSEM 11014	片段重组期间生存时间已过期。
IP_PARAM_PROBLEM 11015	参数问题。
IP_SOURCE_QUENCH 11016	数据报到达太快，无法处理，数据报可能已被丢弃。
IP_OPTION_TOO_BIG 11017	IP 选项太大。
IP_BAD_DESTINATION 11018	一个错误的目标。
IP_GENERAL_FAILURE 11050	常规故障。对于某些格式不正确的 ICMP 数据包，可能会返回此错误。

类型: ULONG

往返时间 (以毫秒为单位)。

OfSize

类型: USHORT

回复的数据大小 (以字节为单位)。

Reserved

类型: USHORT

预留给系统使用。

Data

类型: PVOID

指向回复数据的指针。

Options

类型: 结构ip_option_information

答复的 IP 标头中的 IP 选项，采用 [IP_OPTION_INFORMATION](#) 结构的形式。

注解

ICMP_ECHO_REPLY 结构由 [IcmpParseReplies](#) 函数用来返回对 IPv4 回显请求的响应。在 64 位平台上，应使用 [ICMP_ECHO_REPLY32](#) 结构。

对于 IPv4，**状态** 成员的某些可能值在 [RFC 792](#) 中指定。

[GetIpErrorMessage](#) 函数可用于检索**状态**成员中**IP_STATUS**错误代码的 IP 帮助程序错误字符串。

ICMP_ECHO_REPLY 结构在 *lpeexport.h* 头文件中定义，该文件自动包含在 *iphlpapi.h* 头文件中。永远不应直接使用 *lpeexport.h* 头文件。

要求

最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[GetIpErrorMessage](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IPAddr](#)

[IP_OPTION_INFORMATION](#)

[IP_OPTION_INFORMATION32](#)

[IcmpCloseHandle](#)

[IcmpCreateFile](#)

[IcmpParseReplies](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

[IcmpSendEcho2Ex](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

ICMP_ERROR_INFO 结构 (ws2ipdef.h)

项目2023/08/28

用于存储收到的 ICMP 错误信息。

语法

C++

```
typedef struct icmp_error_info {
    SOCKADDR_INET srcaddress;
    IPPROTO      protocol;
    UINT8        type;
    UINT8        code;
} ICMP_ERROR_INFO, *PICMP_ERROR_INFO;
```

成员

srcaddress

类型: [SOCKADDR_INET](#)

ICMP 错误的源 IP 地址。

protocol

类型: [IPPROTO](#)

ICMP 错误的协议 (IPPROTO_ICMP 或 IPPROTO_ICMPV6)。

type

类型: [UINT8](#)

ICMP 错误类型。

code

类型: [UINT8](#)

ICMP 错误代码。

要求

最低受支持的客户端	Windows 10 版本 2004 (10.0; 内部版本 19041)
最低受支持的服务器	Windows Server 版本 2004 (10.0; 内部版本 19041)
标头	ws2ipdef.h (包括 ws2tcpip.h)

反馈

此页面是否有帮助?



[在 Microsoft Q&A 获取帮助](#)

ICMP_ECHO_REPLY32 结构 (ipexport.h)

项目2023/08/24

ICMP_ECHO_REPLY32 结构描述为响应 64 位平台上的 IPv4 回显请求而返回的数据。

语法

C++

```
typedef struct icmp_echo_reply32 {
    IPAddr Address;
    ULONG Status;
    ULONG RoundTripTime;
    USHORT DataSize;
    USHORT Reserved;
    VOID POINTER_32 *Data;
    struct ip_option_information32 Options;
} ICMP_ECHO_REPLY32, *PICMP_ECHO_REPLY32;
```

成员

Address

类型: IPAddr

回复 IPv4 地址，采用 IPAddr 结构的形式。

Status

类型: ULONG

回显请求的状态，采用 IP_STATUS 代码的形式。此成员的可能值在 *Ipexport.h* 头文件中定义。

值	含义
IP_SUCCESS 0	状态为成功。
IP_BUF_TOO_SMALL 11001	回复缓冲区太小。
IP_DEST_NET_UNREACHABLE 11002	无法访问目标网络。

IP_DEST_HOST_UNREACHABLE	无法访问目标主机。
11003	
IP_DEST_PROT_UNREACHABLE	无法访问目标协议。
11004	
IP_DEST_PORT_UNREACHABLE	无法访问目标端口。
11005	
IP_NO_RESOURCES	可用 IP 资源不足。
11006	
IP_BAD_OPTION	指定了错误的 IP 选项。
11007	
IP_HW_ERROR	发生了硬件错误。
11008	
IP_PACKET_TOO_BIG	数据包太大。
11009	
IP_REQ_TIMED_OUT	请求超时。
11010	
IP_BAD_REQ	错误的请求。
11011	
IP_BAD_ROUTE	一条糟糕的路线。
11012	
IP_TTL_EXPIRED_TRANSIT	(TTL 的生存时间) 在传输中过期。
11013	
IP_TTL_EXPIRED_REASSEM	片段重组期间生存时间已过期。
11014	
IP_PARAM_PROBLEM	参数问题。
11015	
IP_SOURCE_QUENCH	数据报到达太快，无法处理，数据报可能已被丢弃。
11016	
IP_OPTION_TOO_BIG	IP 选项太大。
11017	
IP_BAD_DESTINATION	一个错误的目标。
11018	
IP_GENERAL_FAILURE	常规故障。对于某些格式不正确的 ICMP 数据包，可能会返回此错误。
11050	

类型: ULONG

往返时间 (以毫秒为单位)。

DataSize

类型: USHORT

回复的数据大小 (以字节为单位)。

Reserved

类型: USHORT

预留给系统使用。

Data

类型: VOID * POINTER_32

指向回复数据的指针。

Options

类型: struct ip_option_information32

回复的 IP 标头中的 IP 选项，采用 [IP_OPTION_INFORMATION32](#) 结构的形式。

注解

ICMP_ECHO_REPLY32 结构由 [IcmpParseReplies](#) 函数用于在 64 位平台上返回对 IPv4 回显请求的响应。仅当定义了 _WIN64 时才定义此结构。

对于 IPv4，**状态** 成员的某些可能值在 [RFC 792](#) 中指定。

[GetIpErrorMessage](#) 函数可用于检索**状态**成员中**IP_STATUS**错误代码的 IP 帮助程序错误字符串。

ICMP_ECHO_REPLY32 结构在 *Ipexport.h* 头文件中定义，该文件自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Ipexport.h* 头文件。

要求

最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[GetIpErrorMessage](#)

[ICMP_ECHO_REPLY](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IPAddr](#)

[IP_OPTION_INFORMATION](#)

[IP_OPTION_INFORMATION32](#)

[IcmpCloseHandle](#)

[IcmpCreateFile](#)

[IcmpParseReplies](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

[Icmpsendecho2Ex](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

ICMPV6_ECHO_REPLY_LH 结构 (ipexport.h)

项目2023/08/24

ICMPV6_ECHO_REPLY结构描述为响应 IPv6 回显请求而返回的数据。

语法

C++

```
typedef struct icmpv6_echo_reply_lh {
    IPV6_ADDRESS_EX Address;
    ULONG Status;
    unsigned int RoundTripTime;
} ICMPV6_ECHO_REPLY_LH, *PICMPV6_ECHO_REPLY_LH;
```

成员

Address

类型: IPV6_ADDRESS_EX

答复的 IPv6 地址，采用 IPV6_ADDRESS_EX 结构的形式。

Status

类型: ULONG

回显请求的状态，采用 IP_STATUS 代码的形式。此成员的可能值在 *Ipexport.h* 头文件中定义。

值	含义
IP_SUCCESS 0	状态为“成功”。
IP_BUF_TOO_SMALL 11001	回复缓冲区太小。
IP_DEST_NET_UNREACHABLE 11002	无法访问目标网络。在 IPv6 术语中，此状态值也定义为 IP_DEST_NO_ROUTE。

IP_DEST_HOST_UNREACHABLE 11003	无法访问目标主机。在 IPv6 术语中，此状态值也定义为 IP_DEST_ADDR_UNREACHABLE。
IP_DEST_PROT_UNREACHABLE 11004	无法访问目标协议。在 IPv6 术语中，此状态值也定义为 IP_DEST_PROHIBITED。
IP_DEST_PORT_UNREACHABLE 11005	无法访问目标端口。
IP_NO_RESOURCES 11006	可用 IP 资源不足。
IP_BAD_OPTION 11007	指定了错误的 IP 选项。
IP_HW_ERROR 11008	发生硬件错误。
IP_PACKET_TOO_BIG 11009	数据包太大。
IP_REQ_TIMED_OUT 11010	请求超时。
IP_BAD_REQ 11011	错误的请求。
IP_BAD_ROUTE 11012	一条糟糕的路线。
IP_TTL_EXPIRED_TRANSIT 11013	传输过程中 IPv6 的跃点限制已过期。在 IPv6 术语中，此状态值也定义为 IP_HOP_LIMIT_EXCEEDED。
IP_TTL_EXPIRED_REASSEM 11014	片段重新组合期间 IPv6 的跃点限制已过期。在 IPv6 术语中，此状态值也定义为 IP_REASSEMBLY_TIME_EXCEEDED。
IP_PARAM_PROBLEM 11015	参数问题。在 IPv6 术语中，此状态值也定义为 IP_PARAMETER_PROBLEM。
IP_SOURCE_QUENCH 11016	数据报到达太快，无法处理，数据报可能已被丢弃。
IP_OPTION_TOO_BIG 11017	IP 选项太大。
IP_BAD_DESTINATION 11018	一个糟糕的目标。
IP_DEST_UNREACHABLE 11040	无法访问目标。

IP_TIME_EXCEEDED 11041	已超出时间。
IP_BAD_HEADER 11042	遇到错误的 IP 标头。
IP_UNRECOGNIZED_NEXT_HEADER 11043	遇到无法识别的下一个标头。
IP_ICMP_ERROR 11044	发生 ICMP 错误。
IP_DEST_SCOPE_MISMATCH 11045	目标范围 ID 不匹配。
IP_GENERAL_FAILURE 11050	常规故障。对于某些格式不正确的 ICMP 数据包，可能会返回此错误。

RoundTripTime

类型: `unsigned int`

往返时间（以毫秒为单位）。

注解

ICMPV6_ECHO_REPLY 结构由 [Icmp6ParseReplies](#) 函数用来返回对 IPv6 回显请求的响应。包含来自 ICMPV6 响应的消息正文的回复数据遵循内存中的 ICMPV6_ECHO_REPLY 结构。

对于 IPv6，**状态** 成员的某些可能值在 RFC 2163 中指定。有关详细信息，请参阅 www.ietf.org/rfc/rfc2463.txt。

[GetIpErrorMessage](#) 函数可用于检索**状态**成员中IP_STATUS错误代码的 IP 帮助程序错误字符串。

ICMPV6_ECHO_REPLY 结构在 Microsoft Windows 软件开发工具包 (SDK) 中包含的公共头文件中定义，但此结构由 Windows XP 和更高版本的 [Icmp6ParseReplies](#) 函数使用。

在 Windows SDK 中，如果目标平台是 Windows XP，则编译应用程序时定义 ICMPV6_ECHO_REPLY_LH 结构，并且以后 (`NTDDI_VERSION >= NTDDI_XP`、`_WIN32_WINNT >= 0x0501` 或 `WINVER >= 0x0501`)。ICMPV6_ECHO_REPLY_LH 结构的类型为 ICMPV6_ECHO_REPLY 结构。如果目标平台不是 Windows XP 及更高版本，则编译应用程序时，ICMPV6_ECHO_REPLY 结构未定义。

此结构在 *Ipexport.h* 头文件中定义，该文件自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Ipexport.h* 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[GetIpErrorMessage](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IPV6_ADDRESS_EX](#)

[IP_OPTION_INFORMATION](#)

[Icmp6CreateFile](#)

[Icmp6ParseReplies](#)

[Icmp6SendEcho2](#)

[IcmpCloseHandle](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获得帮助](#)

IN_ADDR 结构 (inaddr.h)

项目2023/08/24

in_addr结构表示 IPv4 地址。

注意 IP 帮助程序中的 IPAddr 类型定义也表示 IPv4 地址，可以在需要时转换为可互换 in_addr 结构。IP 帮助程序中的 in_addr 结构与 Windows 套接字 in_addr 结构具有相同的语法和用法，可与 Windows 套接字中使用的 in_addr 结构互换。Windows 套接字还为 in_addr 结构定义 IN_ADDR typedef。

语法

C++

```
typedef struct in_addr {
    union {
        struct {
            UCHAR s_b1;
            UCHAR s_b2;
            UCHAR s_b3;
            UCHAR s_b4;
        } S_un_b;
        struct {
            USHORT s_w1;
            USHORT s_w2;
        } S_un_w;
        ULONG S_addr;
    } S_un;
} IN_ADDR, *PIN_ADDR, *LPIN_ADDR;
```

成员

S_un

S_un.S_un_b

主机的 IPv4 地址格式为 4 u_char。

S_un.S_un_b.s_b1

S_un.S_un_b.s_b2

`S_un.S_un_b.s_b3`

`S_un.S_un_b.s_b4`

`S_un.S_un_w`

格式化为两 `u_short` 的主机的 IPv4 地址。

`S_un.S_un_w.s_w1`

`S_un.S_un_w.s_w2`

`S_un.S_addr`

格式化为 `u_long` 的主机的地址。

注解

`IPAddr` 类型定义还表示 IPv4 地址，可以在需要时转换为 `in_addr` 结构。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 中，头文件的组织已更改，`in_addr` 结构在 `Ipexport.h` 头文件自动包含的 `Inaddr.h` 头文件中定义。在为 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上，`in_addr` 结构在 `Ipexport.h` 头文件中声明。

要求

最低受支持的客户端	
	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	
	Windows 2000 Server [仅限桌面应用]
标头	
	inaddr.h (包括 Ipexport.h)

另请参阅

[ARP_SEND_REPLY](#)

[AddIPAddress](#)

[GetBestInterface](#)

[GetRTTAndHopCount](#)

[ICMP_ECHO_REPLY](#)

[IP_UNIDIRECTIONAL_ADAPTER_ADDRESS](#)

[IcmpSendEcho](#)

[IcmpSendEcho2](#)

[SendARP](#)

[in_addr \(Winsock\)](#)

反馈

此页面是否有帮助？

[!\[\]\(14819fa071d42f43a37703c4efcfdbcc_img.jpg\) 是](#)

[!\[\]\(6a4dfad1c9bd36ac899168f5177b208b_img.jpg\) 否](#)

[在 Microsoft Q&A 获得帮助](#)

INTERFACE_HARDWARE_TIMESTAMP_C APABILITIES 结构 (iphlpapi.h)

项目2023/08/24

介绍网络接口卡 (NIC) 硬件的时间戳功能。

有关详细信息和代码示例，请参阅 [数据包时间戳](#)。

语法

C++

```
typedef struct _INTERFACE_HARDWARE_TIMESTAMP_CAPABILITIES {
    BOOLEAN PtpV2OverUdpIPv4EventMessageReceive;
    BOOLEAN PtpV2OverUdpIPv4AllMessageReceive;
    BOOLEAN PtpV2OverUdpIPv4EventMessageTransmit;
    BOOLEAN PtpV2OverUdpIPv4AllMessageTransmit;
    BOOLEAN PtpV2OverUdpIPv6EventMessageReceive;
    BOOLEAN PtpV2OverUdpIPv6AllMessageReceive;
    BOOLEAN PtpV2OverUdpIPv6EventMessageTransmit;
    BOOLEAN PtpV2OverUdpIPv6AllMessageTransmit;
    BOOLEAN AllReceive;
    BOOLEAN AllTransmit;
    BOOLEAN TaggedTransmit;
} INTERFACE_HARDWARE_TIMESTAMP_CAPABILITIES,
*PINTERFACE_HARDWARE_TIMESTAMP_CAPABILITIES;
```

成员

PtpV2OverUdpIPv4EventMessageReceive

类型: [BOOLEAN](#)

TRUE 表示在数据包接收期间，NIC 可以在硬件中识别 IPv4 UDP 数据包中包含的 PTP 版本 2 事件消息，并且可以在硬件中生成与接收此类数据包时对应的时间戳。值为 FALSE 表示硬件无法执行此操作。

PtpV2OverUdpIPv4AllMessageReceive

类型: [BOOLEAN](#)

TRUE 表示在数据包接收期间，NIC 可以在硬件中识别任何 PTP 版本 2 消息 (而不仅仅是 IPv4 UDP 数据包中包含的 PTP 事件消息)，并且可以在硬件中生成对应于接收此类数据

包的时间戳。 值为 FALSE 表示硬件无法执行此操作。

PtpV2OverUdpIPv4EventMessageTransmit

类型: **BOOLEAN**

TRUE 表示在数据包传输期间, NIC 可以在硬件中识别 IPv4 UDP 数据包中包含的 PTP 版本 2 事件消息, 并且可以在硬件中生成与传输此类数据包时对应的时间戳。 值为 FALSE 表示硬件无法执行此操作。

PtpV2OverUdpIPv4AllMessageTransmit

类型: **BOOLEAN**

TRUE 表示在数据包传输期间, NIC 可以在硬件中识别任何 PTP 版本 2 消息(而不仅仅是 IPv4 UDP 数据包中包含的 PTP 事件消息), 并且可以在硬件中生成与传输此类数据包时对应的时间戳。 值为 FALSE 表示硬件无法执行此操作。

PtpV2OverUdpIPv6EventMessageReceive

类型: **BOOLEAN**

与 *PtpV2OverUdpIPv4EventMsgReceiveHw* 相同, 只不过它适用于 IPv6。

PtpV2OverUdpIPv6AllMessageReceive

类型: **BOOLEAN**

与 *PtpV2OverUdpIPv4AllMsgReceiveHw* 相同, 只不过它适用于 IPv6。

PtpV2OverUdpIPv6EventMessageTransmit

类型: **BOOLEAN**

与 *PtpV2OverUdpIPv4EventMsgTransmitHw* 相同, 只不过它适用于 IPv6。

PtpV2OverUdpIPv6AllMessageTransmit

类型: **BOOLEAN**

与 *PtpV2OverUdpIPv4AllMsgTransmitHw* 相同, 只不过它适用于 IPv6。

AllReceive

类型: **BOOLEAN**

TRUE 表示 NIC 可以为接收的所有数据包生成硬件时间戳, (即, 而不仅仅是 PTP)。 值为 FALSE 表示硬件无法执行此操作。

AllTransmit

类型: **BOOLEAN**

TRUE 表示 NIC 可以为所有传输的数据包生成硬件时间戳, (, 而不只是 PTP)。 值为 FALSE 表示硬件无法执行此操作。

TaggedTransmit

类型: **BOOLEAN**

TRUE 指示 NIC 可以在应用程序指示这样做时为任何特定传输的数据包生成硬件时间戳。 值为 FALSE 表示硬件无法执行此操作。 请参阅 [TIMESTAMPING_CONFIG](#) (和 [TIMESTAMPING_FLAG_TX](#)) , 以确定如何在通过 [Windows 套接字](#)发送 UDP 数据包时请求时间戳。

注解

INTERFACE_HARDWARE_TIMESTAMP_CAPABILITIES 结构的所有成员都表示硬件时间戳功能。 硬件时间戳是使用 NIC 的硬件时钟生成的。

不支持同时启用硬件和软件时间戳。

要求

最低受支持的客户端	Windows 10内部版本 20348
最低受支持的服务器	Windows 10内部版本 20348
标头	iphlpapi.h

另请参阅

- [包时间戳](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

INTERFACE_HARDWARE_CROSSTIMESTAMP 结构 (iphlpapi.h)

项目2023/08/24

描述从网络适配器检索到的交叉时间戳。 交叉时间戳是指一组网络接口卡 (NIC,) 硬件时间戳和系统时间戳 () 彼此非常接近。

若要检索交叉时间戳，请调用 [CaptureInterfaceHardwareCrossTimestamp](#) 函数。 该函数以 INTERFACE_HARDWARE_CROSSTIMESTAMP 对象的形式从网络适配器返回时间戳。

有关详细信息和代码示例，请参阅 [数据包时间戳](#)。

语法

C++

```
typedef struct _INTERFACE_HARDWARE_CROSSTIMESTAMP {
    ULONG64 SystemTimestamp1;
    ULONG64 HardwareClockTimestamp;
    ULONG64 SystemTimestamp2;
} INTERFACE_HARDWARE_CROSSTIMESTAMP, *PINTERFACE_HARDWARE_CROSSTIMESTAMP;
```

成员

SystemTimestamp1

类型: [ULONG64](#)

网络适配器驱动程序使用系统时间戳填充此内容，该时间戳的值对应于 [QueryPerformanceCounter](#) (QPC) 返回的值。

在 HardwareClockTimestamp 之前获取 SystemTimestamp1;而 SystemTimestamp2 位于 HardwareClockTimestamp 之后。 时间戳值尽可能彼此接近。

HardwareClockTimestamp

类型: [ULONG64](#)

网络适配器驱动程序使用从其网络接口卡 (NIC) 硬件时钟获取的值来填充此值。

SystemTimestamp2

类型: **ULONG64**

网络适配器驱动程序使用系统时间戳填充此内容，该时间戳的值对应于 [QueryPerformanceCounter](#) (QPC) 返回的值。

在 *HardwareClockTimestamp* 之前获取 *SystemTimestamp1*;而 *SystemTimestamp2* 位于 *HardwareClockTimestamp* 之后。 时间戳值尽可能彼此接近。

要求

最低受支持的客户端	Windows 11 (内部版本 10.0.22000.194)
最低受支持的服务器	Windows Server 2022
标头	iphlpapi.h

另请参阅

[包时间戳](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

INTERFACE_SOFTWARE_TIMESTAMP_CAPABILITIES 结构 (iphlpapi.h)

项目2023/08/24

介绍 NIC 微型端口驱动程序的软件时间戳功能。

有关详细信息和代码示例，请参阅 [数据包时间戳](#)。

语法

C++

```
typedef struct _INTERFACE_SOFTWARE_TIMESTAMP_CAPABILITIES {
    BOOLEAN AllReceive;
    BOOLEAN AllTransmit;
    BOOLEAN TaggedTransmit;
} INTERFACE_SOFTWARE_TIMESTAMP_CAPABILITIES,
*PINTERFACE_SOFTWARE_TIMESTAMP_CAPABILITIES;
```

成员

AllReceive

类型: [BOOLEAN](#)

还包含描述 NIC 微型端口驱动程序的软件时间戳功能的成员。不是硬件功能。TRUE 表示 NIC 的微型端口驱动程序可以为所有收到的数据包生成软件时间戳。值为 FALSE 表示软件无法执行此操作。

AllTransmit

类型: [BOOLEAN](#)

不是硬件功能。类似于 AllReceiveSw，但它适用于传输方向。TRUE 表示 NIC 的微型端口驱动程序可以为所有传输的数据包生成软件时间戳。值为 FALSE 表示软件无法执行此操作。

TaggedTransmit

类型: [BOOLEAN](#)

不是硬件功能。 TRUE 指示 NIC 的微型端口驱动程序可以在应用程序指示时为任何特定传输的数据包生成软件时间戳。 值为 FALSE 表示软件无法执行此操作。 请参阅 [TIMESTAMPING_CONFIG](#) (和 [TIMESTAMPING_FLAG_TX](#))，以确定如何在通过 [Windows 套接字](#)发送 UDP 数据包时请求时间戳。

注解

`INTERFACE_SOFTWARE_TIMESTAMP_CAPABILITIES` 结构的所有成员都表示软件时间戳功能。 NIC 驱动程序生成的软件时间戳对应于通过调用 [QueryPerformanceCounter](#) 获取的计数器值。

不支持同时启用硬件和软件时间戳。

要求

最低受支持的客户端	Windows 10内部版本 20348
最低受支持的服务器	Windows 10内部版本 20348
标头	iphlpapi.h

另请参阅

- [包时间戳](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

INTERFACE_TIMESTAMP_CAPABILITIES 结构 (iphlpapi.h)

项目2023/08/24

描述网络适配器支持的确切时间戳功能。

若要检索网络适配器支持的时间戳功能，请调用 [GetInterfaceSupportedTimestampCapabilities](#) 函数。该函数以 INTERFACE_TIMESTAMP_CAPABILITIES 对象的形式返回支持的时间戳功能。

有关详细信息和代码示例，请参阅[数据包时间戳](#)。

语法

C++

```
typedef struct _INTERFACE_TIMESTAMP_CAPABILITIES {
    ULONG64                               HardwareClockFrequencyHz;
    BOOLEAN                                SupportsCrossTimestamp;
    INTERFACE_HARDWARE_TIMESTAMP_CAPABILITIES HardwareCapabilities;
    INTERFACE_SOFTWARE_TIMESTAMP_CAPABILITIES SoftwareCapabilities;
} INTERFACE_TIMESTAMP_CAPABILITIES, *PINTERFACE_TIMESTAMP_CAPABILITIES;
```

成员

HardwareClockFrequencyHz

类型: [ULONG64](#)

包含网络适配器的硬件时钟的频率，舍入到以 Hertz 为单位的最接近的整数。请注意，这是标称频率，实际频率可能不相同。此数据可用于向最终用户显示名义时钟频率，仅供参考。*HardwareClockFrequencyHz* 可以包含值 0。

SupportsCrossTimestamp

类型: [BOOLEAN](#)

值为 TRUE 表示网络适配器驱动程序能够生成硬件跨时间戳。交叉时间戳是指一组网络接口，卡 (NIC) 硬件时间戳和系统时间戳彼此非常接近。值为 FALSE 表示此功能不存在。

HardwareCapabilities

类型: [INTERFACE_HARDWARE_TIMESTAMP_CAPABILITIES](#)

介绍网络接口卡 (NIC) 硬件的时间戳功能。 不支持同时启用硬件和软件时间戳。

SoftwareCapabilities

类型: [INTERFACE_SOFTWARE_TIMESTAMP_CAPABILITIES](#)

介绍网络接口卡 (NIC 的) 微型端口驱动程序的软件时间戳功能。 不支持同时启用硬件和软件时间戳。

要求

最低受支持的客户端	Windows 11 (内部版本 10.0.22000.194)
最低受支持的服务器	Windows Server 2022
标头	iphlpapi.h

另请参阅

[包时间戳](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

(iptypes.h) IP_ADAPTER_ADDRESSES_LH 结构

项目2023/08/24

IP_ADAPTER_ADDRESSES结构是特定适配器地址链接列表的标头节点。此结构可以同时用作IP_ADAPTER_ADDRESSES结构链接列表的一部分。

语法

C++

```
typedef struct _IP_ADAPTER_ADDRESSES_LH {
    union {
        ULLONG Alignment;
        struct {
            ULONG Length;
            IF_INDEX IfIndex;
        };
    };
    struct _IP_ADAPTER_ADDRESSES_LH *Next;
    PCHAR AdapterName;
    PIP_ADAPTER_UNICAST_ADDRESS_LH FirstUnicastAddress;
    PIP_ADAPTER_ANYCAST_ADDRESS_XP FirstAnycastAddress;
    PIP_ADAPTER_MULTICAST_ADDRESS_XP FirstMulticastAddress;
    PIP_ADAPTER_DNS_SERVER_ADDRESS_XP FirstDnsServerAddress;
    PWCHAR DnsSuffix;
    PWCHAR Description;
    PWCHAR FriendlyName;
    BYTE PhysicalAddress[MAX_ADAPTER_ADDRESS_LENGTH];
    ULONG PhysicalAddressLength;
    union {
        ULONG Flags;
        struct {
            ULONG DdnsEnabled : 1;
            ULONG RegisterAdapterSuffix : 1;
            ULONG Dhcpv4Enabled : 1;
            ULONG ReceiveOnly : 1;
            ULONG NoMulticast : 1;
            ULONG Ipv6OtherStatefulConfig : 1;
            ULONG NetbiosOverTcpipEnabled : 1;
            ULONG Ipv4Enabled : 1;
            ULONG Ipv6Enabled : 1;
            ULONG Ipv6ManagedAddressConfigurationSupported : 1;
        };
    };
    ULONG Mtu;
    IFTYPE IfType;
```

```

IF_OPER_STATUS           OperStatus;
IF_INDEX                Ipv6IfIndex;
ULONG                  ZoneIndices[16];
PIP_ADAPTER_PREFIX_XP   FirstPrefix;
ULONG64                 TransmitLinkSpeed;
ULONG64                 ReceiveLinkSpeed;
PIP_ADAPTER_WINS_SERVER_ADDRESS_LH FirstWinsServerAddress;
PIP_ADAPTER_GATEWAY_ADDRESS_LH    FirstGatewayAddress;
ULONG                  Ipv4Metric;
ULONG                  Ipv6Metric;
IF_LUID                 Luid;
SOCKET_ADDRESS          Dhcpv4Server;
NET_IF_COMPARTMENT_ID   CompartmentId;
NET_IF_NETWORK_GUID     NetworkGuid;
NET_IF_CONNECTION_TYPE  ConnectionType;
TUNNEL_TYPE              TunnelType;
SOCKET_ADDRESS          Dhcpv6Server;
BYTE                   Dhcpv6ClientDuid[MAX_DHCPV6_DUID_LENGTH];
ULONG                  Dhcpv6ClientDuidLength;
ULONG                  Dhcpv6Iaid;
PIP_ADAPTER_DNS_SUFFIX  FirstDnsSuffix;
} IP_ADAPTER_ADDRESSES_LH, *PIP_ADAPTER_ADDRESSES_LH;

```

成员

Alignment

类型: **ULONGLONG**

保留。由编译器用于对齐结构。

Length

类型: **ULONG**

此结构的长度（以字节为单位）。请注意，windows XP SP1 及更高版本以及 Windows Vista 及更高版本上IP_ADAPTER_ADDRESSES结构的长度已更改。

IfIndex

类型: **DWORD**

与这些地址关联的 IPv4 接口的索引。在 Windows Server 2003 和 Windows XP 上，如果接口上没有可用的 IPv4，则此成员为零。

Next

类型: **struct _IP_ADAPTER_ADDRESSES***

指向列表中下一个适配器的指针地址结构。

AdapterName

类型: **PCHAR**

包含与这些地址关联的适配器名称的字符数组。 与适配器的友好名称不同,
AdapterName 中指定的适配器名称是永久性的, 不能由用户修改。

FirstUnicastAddress

类型: **PIP_ADAPTER_UNICAST_ADDRESS**

指向适配器 IP 单播地址链接列表中的第一个 [IP_ADAPTER_UNICAST_ADDRESS](#) 结构的指针。

FirstAnycastAddress

类型: **PIP_ADAPTER_ANYCAST_ADDRESS**

指向适配器 IP 任意播地址链接列表中的第一个 [IP_ADAPTER_ANYCAST_ADDRESS](#) 结构的指针。

FirstMulticastAddress

类型: **PIP_ADAPTER_MULTICAST_ADDRESS**

指向适配器 IP 多播地址列表中第一个 [IP_ADAPTER_MULTICAST_ADDRESS](#) 结构的指针。

FirstDnsServerAddress

类型: **PIP_ADAPTER_DNS_SERVER_ADDRESS**

指向适配器 DNS 服务器地址链接列表中的第一个 [IP_ADAPTER_DNS_SERVER_ADDRESS](#) 结构的指针。

DnsSuffix

类型: **PWCHAR**

域名系统 (与此适配器关联的 DNS) 后缀。

Description

类型: **PWCHAR**

适配器的说明。 此成员是只读的。

`FriendlyName`

类型: `PWCHAR`

适配器的用户友好名称。例如：“本地连接 1”。此名称显示在 `ipconfig` 命令行程序和 Connection 文件夹等上下文中。此成员是只读的，不能使用任何 IP 帮助程序函数进行修改。

此成员是 NDIS 使用的 `ifAlias` 字段，如 [RFC 2863](#) 中所述。安装 NDIS 驱动程序时，NDIS 接口提供程序可以设置 `ifAlias` 字段。对于 NDIS 微型端口驱动程序，此字段由 NDIS 设置。

`PhysicalAddress[MAX_ADAPTER_ADDRESS_LENGTH]`

类型: `BYTE[MAX_ADAPTER_ADDRESS_LENGTH]`

适配器的媒体访问控制 (MAC) 地址。例如，在以太网网络上，此成员将指定以太网硬件地址。

`PhysicalAddressLength`

类型: `DWORD`

`PhysicalAddress` 成员中指定的地址的长度（以字节为单位）。对于没有数据链接层的接口，此值为零。

`Flags`

类型: `DWORD`

一组为适配器指定各种设置的标志。这些值在 `Iptypes.h` 头文件中定义。可以组合这些标志位。

标志	含义
<code>IP_ADAPTER_DDNS_ENABLED</code> 0x0001	在此适配器上启用了动态 DNS。
<code>IP_ADAPTER_REGISTER_ADAPTER_SUFFIX</code> 0x0002	注册此适配器的 DNS 后缀。
<code>IP_ADAPTER_DHCP_ENABLED</code> 0x0004	在此适配器上启用了动态主机配置协议 (DHCP)。 。
<code>IP_ADAPTER_RECEIVE_ONLY</code> 0x0008	适配器是仅限接收的适配器。
<code>IP_ADAPTER_NO_MULTICAST</code>	适配器不是多播接收方。

0x0010

IP_ADAPTER_IPV6_OTHER_STATEFUL_CONFIG
0x0020 适配器包含其他 IPv6 特定的有状态配置信息。

IP_ADAPTER_NETBIOS_OVER_TCPIP_ENABLED
0x0040 适配器通过 TCP/IP 为 NetBIOS 启用。

注意 仅当应用程序已针对 NTDDI 版本等于或大于 NTDDI_LONGHORN 的目标平台编译时，Windows Vista 和更高版本才支持此标志。此标志在 IP_ADAPTER_ADDRESSES_LH 结构中定义为 NetbiosOverTcpipEnabled 位域。

IP_ADAPTER_IPV4_ENABLED 为 IPv4 启用适配器。

0x0080

注意 仅当应用程序已针对 NTDDI 版本等于或大于 NTDDI_LONGHORN 的目标平台编译时，Windows Vista 和更高版本才支持此标志。此标志在 IP_ADAPTER_ADDRESSES_LH 结构中定义为 Ipv4Enabled 位域。

IP_ADAPTER_IPV6_ENABLED 为 IPv6 启用适配器。

0x0100

注意 仅当应用程序已针对 NTDDI 版本等于或大于 NTDDI_LONGHORN 的目标平台编译时，Windows Vista 和更高版本才支持此标志。此标志在 IP_ADAPTER_ADDRESSES_LH 结构中定义为 Ipv6Enabled 位域。

IP_ADAPTER_IPV6_MANAGE_ADDRESS_CONFIG 为 IPv6 托管地址配置启用适配器。

0x0200

注意 仅当应用程序已针对 NTDDI 版本等于或大于 NTDDI_LONGHORN 的目标平台编译时，Windows Vista 和更高版本才支持此标志。此标志

在 IP_ADAPTER_ADDRESSES_LH 结构中定义为
Ipv6ManagedAddressConfigura
nSupported 位域。

DdnsEnabled

RegisterAdapterSuffix

Dhcpv4Enabled

ReceiveOnly

NoMulticast

Ipv6OtherStatefulConfig

NetbiosOverTcpipEnabled

Ipv4Enabled

Ipv6Enabled

Ipv6ManagedAddressConfigurationSupported

Mtu

类型: DWORD

最大传输单位 (MTU) 大小 (以字节为单位)。

IfType

类型: DWORD

Internet 分配名称机构 (IANA) 定义的接口类型。 *Ipifcons.h* 头文件中列出了接口类型的可能值。

下表列出了接口类型的常见值，尽管可能还有许多其他值。

值	含义
IF_TYPE_OTHER 1	其他一些类型的网络接口。
IF_TYPE_ETHERNET_CSMACD	以太网网络接口。

IF_TYPE_ISO88025_TOKENRING 9	令牌环网络接口。
IF_TYPE_PPP 23	PPP 网络接口。
IF_TYPE_SOFTWARE_LOOPBACK 24	软件环回网络接口。
IF_TYPE_ATM 37	ATM 网络接口。
IF_TYPE_IEEE80211 71	<p>IEEE 802.11 无线网络接口。 在 Windows Vista 及更高版本中，无线网卡报告为 IF_TYPE_IEEE80211。在早期版本的 Windows 上，无线网卡报告为 IF_TYPE_ETHERNET_CSMACD。</p> <p>在装有 SP3 的 Windows XP 和安装了 SP2 的 Windows XP 的 Windows XP 上的 SP2 x86 上，WlanEnumInterfaces 函数可用于枚举本地计算机上的无线接口。</p>
IF_TYPE_TUNNEL 131	隧道类型封装网络接口。
IF_TYPE_IEEE1394 144	IEEE 1394 (Firewire) 高性能串行总线网络接口。

OperStatus

类型: IF_OPER_STATUS

RFC 2863 中定义的接口的操作状态。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2863.txt>。此成员可以是 *iftypes.h* 头文件中定义的 IF_OPER_STATUS 枚举类型的值之一。在 Windows Vista 及更高版本上，头文件已重新组织，此枚举在 *Ifdef.h* 头文件中定义。

值	含义
IfOperStatusUp 1	接口已启动，并且能够传递数据包。
IfOperStatusDown 2	<p>接口已关闭，并且不处于传递数据包的条件。 IfOperStatusDown 状态有两种含义，具体取决于 AdminStatus 成员的值。如果 AdminStatus 未设置为 NET_IF_ADMIN_STATUS_DOWN 并且 OperStatus 设置为 IfOperStatusDown，则假定接口上存在错误条件。如果 AdminStatus 设置为 IfOperStatusDown，则 ifOperStatus 通常也会设置为 IfOperStatusDown 或</p>

	IfOperStatusNotPresent , 并且接口上不一定有故障条件。
IfOperStatusTesting 3	接口处于测试模式。
IfOperStatusUnknown 4	接口的操作状态未知。
IfOperStatusDormant 5	接口实际上并不处于传递数据包的条件, (它未启动), 而是处于挂起状态, 正在等待某些外部事件。对于按需接口, 此新状态标识接口等待事件将其置于 IfOperStatusUp 状态的情况。
IfOperStatusNotPresent 6	IfOperStatusDown 状态的优化, 指示相关接口已关闭, 特别是因为某些组件 (通常托管系统中不存在硬件组件)。
IfOperStatusLowerLayerDown 7	IfOperStatusDown 状态的优化。此新状态指示此接口在一个或多个其他接口上运行, 并且此接口已关闭, 特别是因为这些较低层接口中的一个或多个已关闭。

Ipv6IfIndex

类型: DWORD

IPv6 IP 地址的接口索引。如果 IPv6 在接口上不可用, 则此成员为零。

注意 此结构成员仅在具有 SP1 及更高版本的 Windows XP 上可用。

ZoneIndices[16]

类型: DWORD[16]

用于组合 [sockaddr](#) 结构的每个范围级别的范围 ID 数组。 [SCOPE_LEVEL](#) 枚举用于为数组编制索引。在 IPv6 上, 可能会根据范围 ID 为单个接口分配多个 IPv6 多播地址。

注意 此结构成员仅在具有 SP1 及更高版本的 Windows XP 上可用。

FirstPrefix

类型: [PIP_ADAPTER_PREFIX](#)

指向适配器的 IP 适配器前缀链接列表中的第一个 [IP_ADAPTER_PREFIX](#) 结构的指针。

注意 此结构成员仅在具有 SP1 及更高版本的 Windows XP 上可用。

TransmitLinkSpeed

类型: **ULONG64**

适配器的传输链路的当前速度 (以位/秒为单位)。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

ReceiveLinkSpeed

类型: **ULONG64**

适配器的接收链接的当前速度 (以位/秒为单位)。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

FirstWinsServerAddress

类型: **PIP_ADAPTER_WINS_SERVER_ADDRESS_LH**

指向 Windows Internet 名称服务链接列表中的第一个
[IP_ADAPTER_WINS_SERVER_ADDRESS](#) 结构的指针, (适配器的 WINS) 服务器地址。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

FirstGatewayAddress

类型: **PIP_ADAPTER_GATEWAY_ADDRESS_LH**

指向适配器网关链接列表中的第一个 [IP_ADAPTER_GATEWAY_ADDRESS](#) 结构的指针。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

Ipv4Metric

类型: **ULONG**

适配器地址的 IPv4 接口指标。此成员仅适用于 IPv4 适配器地址。

用于计算 IPv4 路由首选项的实际路由指标是[MIB_IPFORWARD_ROW2](#)结构的 Metric 成员中指定的路由指标偏移量的总和，以及此成员中为 IPv4 指定的接口指标。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

Ipv6Metric

类型: **ULONG**

适配器地址的 IPv6 接口指标。此成员仅适用于 IPv6 适配器地址。

用于计算 IPv6 路由首选项的实际路由指标是[MIB_IPFORWARD_ROW2](#)结构的 Metric 成员中指定的路由指标偏移量的总和，以及此成员中为 IPv4 指定的接口指标。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

Luid

类型: **IF_LUID**

适配器地址的接口 LUID。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

Dhcpv4Server

类型: [SOCKET_ADDRESS](#)

适配器地址的 DHCP 服务器的 IPv4 地址。此成员仅适用于使用 DHCP 配置的 IPv4 适配器地址。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

CompartmentId

类型: **NET_IF_COMPARTMENT_ID**

适配器地址的路由隔离舱 ID。

注意 此结构成员仅适用于 Windows Vista 及更高版本。此成员当前不受支持，保留供将来使用。

NetworkGuid

类型: **NET_IF_NETWORK_GUID**

与接口所属的网络关联的 GUID。

如果接口提供程序无法提供网络 GUID，则此成员可以是零 GUID。在这种情况下，接口由 NDIS 在默认网络中注册。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

ConnectionType

类型: **NET_IF_CONNECTION_TYPE**

适配器地址的接口连接类型。

此成员可以是 *Ifdef.h* 头文件中定义的 **NET_IF_CONNECTION_TYPE** 枚举类型的值之一。

值	含义
NET_IF_CONNECTION_DEDICATED 1	连接类型是专用的。当媒体感知为 TRUE 时，连接会自动启动。例如，以太网连接是专用的。
NET_IF_CONNECTION_PASSIVE 2	连接类型为被动连接。远程端必须启动与本地工作站的连接。例如，RAS 接口是被动的。
NET_IF_CONNECTION_DEMAND 3	连接类型为 demand-dial。此类型的连接是为了响应本地操作 (发送数据包，例如)。
NET_IF_CONNECTION_MAXIMUM 4	NET_IF_CONNECTION_TYPE 枚举类型的最大值。这不是 ConnectionType 成员的合法值。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

TunnelType

类型: **TUNNEL_TYPE**

如果适配器地址为隧道，则为隧道使用的封装方法。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

隧道类型由 Internet 域名分配机构 (IANA) 定义。有关详细信息，请参阅 <http://www.iana.org/assignments/ianaiftype-mib>。此成员可以是 *Ifdef.h* 头文件中定义的 **TUNNEL_TYPE** 枚举类型的值之一。

值	含义
TUNNEL_TYPE_NONE 0	不是隧道。
TUNNEL_TYPE_OTHER 1	以下任何隧道类型都不是。
TUNNEL_TYPE_DIRECT 2	数据包直接封装在普通 IP 标头中，没有中间标头，并单播到远程隧道终结点。
TUNNEL_TYPE_6TO4 11	IPv6 数据包直接封装在 IPv4 标头中，没有中间标头，并单播到由 6to4 协议确定的目标。
TUNNEL_TYPE_ISATAP 13	IPv6 数据包直接封装在 IPv4 标头中，没有中间标头，并单播到由 ISATAP 协议确定的目标。
TUNNEL_TYPE_TEREDO 14	适用于 IPv6 数据包的 Teredo 封装。
TUNNEL_TYPE_IPHTTPS 15	IPv6 数据包的基于 HTTPS 的 IP 封装。

注意 此枚举值仅适用于 Windows 7、Windows Server 2008 R2 及更高版本。

Dhcpv6Server

类型: SOCKET_ADDRESS

适配器地址的 DHCPv6 服务器的 IPv6 地址。此成员仅适用于使用 DHCPv6 配置的 IPv6 适配器地址。此结构成员当前不受支持，保留供将来使用。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

Dhcpv6ClientDuid[MAX_DHCPV6_DUID_LENGTH]

类型: BYTE[MAX_DHCPV6_DUID_LENGTH]

DHCP 唯一标识符 (DHCPv6 客户端的 DUID)。此成员仅适用于使用 DHCPv6 配置的 IPv6 适配器地址。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

Dhcpv6ClientDuidLength

类型: ULONG

DHCPv6 客户端的 DHCP 唯一标识符 (DUID) 的长度 (以字节为单位)。此成员仅适用于使用 DHCPv6 配置的 IPv6 适配器地址。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

Dhcpv6Iaid

类型: ULONG

DHCPv6 客户端选择的标识关联的标识符。此成员仅适用于使用 DHCPv6 配置的 IPv6 适配器地址。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

FirstDnsSuffix

类型: PIP_ADAPTER_DNS_SUFFIX

指向适配器 DNS 后缀链接列表中的第一个 [IP_ADAPTER_DNS_SUFFIX](#) 结构的指针。

注意 此结构成员仅在 Windows Vista SP1 及更高版本以及 Windows Server 2008 及更高版本上可用。

注解

[GetAdaptersAddresses](#) 函数检索 IPv4 和 IPv6 地址的信息，并将此信息作为 [IP_ADAPTER_ADDRESSES](#) 结构的链接列表返回。

IfIndex 和 **Ipv6IfIndex** 成员中指定的适配器索引值可能会在禁用然后启用适配器时发生更改，或者在其他情况下，不应被视为永久性。

IfType 成员的值在 *Ipfcons.h* 头文件中定义。当前仅支持 **IfType** 成员的说明中列出的可能值。

windows XP SP1 及更高版本上 [IP_ADAPTER_ADDRESSES](#) 结构的大小已更改。Windows Vista 及更高版本上 [IP_ADAPTER_ADDRESSES](#) 结构的大小也发生了变化。在 Windows Vista SP1 及更高版本以及 Windows Server 2008 及更高版本上，[IP_ADAPTER_ADDRESSES](#) 结构的大小也发生了变化。**应使用 Length 成员来确定正在使用哪个版本的 IP_ADAPTER_ADDRESSES 结构。**

Windows XP 上的 [IP_ADAPTER_ADDRESSES](#) 结构版本 SP1 及更高版本添加了以下新成员：**Ipv6IfIndex**、**ZoneIndices** 和 **FirstPrefix**。

Windows Vista 和更高版本上的 [IP_ADAPTER_ADDRESSES](#) 结构版本添加了以下新成员：**TransmitLinkSpeed**、**ReceiveLinkSpeed**、**FirstWinsServerAddress**、**FirstGatewayAddress**、**Ipv4Metric**、**Ipv6Metric**、**Luid**、**Dhcpv4Server**、**CompartmentId**、**NetworkGuid**、**ConnectionType**、**TunnelType**、**Dhcpv6Server**、**Dhcpv6ClientDuid**、**Dhcpv6ClientDuidLength** 和 **Dhcpv6Iaid**。

Windows Vista SP1 及更高版本以及 Windows Server 2008 及更高版本上 [IP_ADAPTER_ADDRESSES](#) 结构的版本添加了以下新成员：**FirstDnsSuffix**。

Ipv4Metric 和 **Ipv6Metric** 成员用于为连接到本地计算机上的多个接口的路由设置路由指标的优先级。

由 [GetAdaptersAddresses](#) 函数返回的 **FirstUnicastAddress** 成员指向的链接 [IP_ADAPTER_UNICAST_ADDRESS](#) 结构的顺序并不反映 IP 地址添加到适配器的顺序，并且可能因 Windows 版本而异。同样，**FirstAnycastAddress** 成员指向的链接 [IP_ADAPTER_ANYCAST_ADDRESS](#) 结构的顺序和 **FirstMulticastAddress** 成员指向的链接

`IP_ADAPTER_MULTICAST_ADDRESS`结构的顺序并不反映 IP 地址添加到适配器的顺序，并且可能因 Windows 版本而异。

此外，由 `FirstUnicastAddress` 成员指向的链接`IP_ADAPTER_UNICAST_ADDRESS`结构和 `FirstPrefix` 成员指向的链接`IP_ADAPTER_PREFIX`结构由操作系统作为单独的内部链接列表进行维护。因此，`FirstUnicastAddress` 成员指向的链接`IP_ADAPTER_UNICAST_ADDRESS`结构的顺序与 `FirstPrefix` 成员所指向的链接`IP_ADAPTER_PREFIX`结构的顺序没有任何关系。

在 Windows Vista 及更高版本上，`FirstPrefix` 成员指向的链接`IP_ADAPTER_PREFIX`结构包括分配给适配器的每个 IP 地址的三个 IP 适配器前缀。其中包括主机 IP 地址前缀、子网 IP 地址前缀和子网广播 IP 地址前缀。此外，每个适配器都有一个多播地址前缀和一个广播地址前缀。

在具有 SP1 及更高版本 Windows Vista 的 Windows XP 上，`FirstPrefix` 成员指向的链接`IP_ADAPTER_PREFIX`结构仅包含分配给适配器的每个 IP 地址的单个 IP 适配器前缀。

在 Windows SDK 中，用于 Windows Vista 及更高版本的结构的版本定义为 `IP_ADAPTER_ADDRESSES_LH`。在 Microsoft Windows 软件开发工具包 (SDK) 中，此结构的版本将用于早期系统（包括 Windows XP SP1 及更高版本）定义为 `IP_ADAPTER_ADDRESSES_XP`。编译应用程序时，如果目标平台是 Windows Vista 且更高版本 (`NTDDI_VERSION >= NTDDI_LONGHORN`、`_WIN32_WINNT >= 0x0600` 或 `WINVER >= 0x0600`)，则 `IP_ADAPTER_ADDRESSES_LH` 结构的类型为 `IP_ADAPTER_ADDRESSES` 结构。如果目标平台不是 Windows Vista 及更高版本，则编译应用程序时，`IP_ADAPTER_ADDRESSES_XP` 结构的类型为 `IP_ADAPTER_ADDRESSES` 结构。

`SOCKET_ADDRESS` 结构在 `IP_ADAPTER_ADDRESSES` 结构中使用。在为 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织已更改，`SOCKET_ADDRESS` 结构在 `Ws2def.h` 头文件中定义，该文件由 `Winsock2.h` 头文件自动包含。在针对 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上，`SOCKET_ADDRESS` 结构在 `Winsock2.h` 头文件中声明。若要使用 `IP_ADAPTER_ADDRESSES` 结构，必须在 `Iphlpapi.h` 头文件之前包含 `Winsock2.h` 头文件。

示例

此示例检索与系统关联的适配器的`IP_ADAPTER_ADDRESSES`结构，并为每个适配器接口打印一些成员。

C++

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#pragma comment(lib, "IPHLPAPI.lib")
```

```
#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int __cdecl main(int argc, char **argv)
{

    /* Declare and initialize variables */

    DWORD dwSize = 0;
    DWORD dwRetVal = 0;

    unsigned int i = 0;

    // Set the flags to pass to GetAdaptersAddresses
    ULONG flags = GAA_FLAG_INCLUDE_PREFIX;

    // default to unspecified address family (both)
    ULONG family = AF_UNSPEC;

    LPVOID lpMsgBuf = NULL;

    PIP_ADAPTER_ADDRESSES pAddresses = NULL;
    ULONG outBufLen = 0;

    PIP_ADAPTER_ADDRESSES pCurrAddresses = NULL;
    PIP_ADAPTER_UNICAST_ADDRESS pUnicast = NULL;
    PIP_ADAPTER_ANYCAST_ADDRESS pAnycast = NULL;
    PIP_ADAPTER_MULTICAST_ADDRESS pMulticast = NULL;
    IP_ADAPTER_DNS_SERVER_ADDRESS *pDnServer = NULL;
    IP_ADAPTER_PREFIX *pPrefix = NULL;

    if (argc != 2) {
        printf(" Usage: getadapteraddresses family\n");
        printf("          getadapteraddresses 4 (for IPv4)\n");
        printf("          getadapteraddresses 6 (for IPv6)\n");
        printf("          getadapteraddresses A (for both IPv4 and IPv6)\n");
        exit(1);
    }

    if (atoi(argv[1]) == 4)
        family = AF_INET;
    else if (atoi(argv[1]) == 6)
        family = AF_INET6;

    outBufLen = sizeof(IP_ADAPTER_ADDRESSES);
    pAddresses = (IP_ADAPTER_ADDRESSES *) MALLOC(outBufLen);

    // Make an initial call to GetAdaptersAddresses to get the
    // size needed into the outBufLen variable
    if (GetAdaptersAddresses(family, flags, NULL, pAddresses, &outBufLen)
        == ERROR_BUFFER_OVERFLOW) {
        FREE(pAddresses);
```

```

    pAddresses = (IP_ADAPTER_ADDRESSES *) MALLOC(outBufLen);
}

if (pAddresses == NULL) {
    printf("Memory allocation failed for IP_ADAPTER_ADDRESSES
struct\n");
    exit(1);
}
// Make a second call to GetAdaptersAddresses to get the
// actual data we want
printf("Memory allocated for GetAdapterAddresses = %d bytes\n",
outBufLen);
printf("Calling GetAdaptersAddresses function with family = ");
if (family == AF_INET)
    printf("AF_INET\n");
if (family == AF_INET6)
    printf("AF_INET6\n");
if (family == AF_UNSPEC)
    printf("AF_UNSPEC\n\n");

dwRetVal =
    GetAdaptersAddresses(family, flags, NULL, pAddresses, &outBufLen);

if (dwRetVal == NO_ERROR) {
    // If successful, output some information from the data we received
    pCurrAddresses = pAddresses;
    while (pCurrAddresses) {
        printf("\tLength of the IP_ADAPTER_ADDRESS struct: %ld\n",
               pCurrAddresses->Length);
        printf("\tIfIndex (IPv4 interface): %u\n", pCurrAddresses-
>IfIndex);
        printf("\tAdapter name: %s\n", pCurrAddresses->AdapterName);

        pUnicast = pCurrAddresses->FirstUnicastAddress;
        if (pUnicast != NULL) {
            for (i = 0; pUnicast != NULL; i++)
                pUnicast = pUnicast->Next;
            printf("\tNumber of Unicast Addresses: %d\n", i);
        } else
            printf("\tNo Unicast Addresses\n");

        pAnycast = pCurrAddresses->FirstAnycastAddress;
        if (pAnycast) {
            for (i = 0; pAnycast != NULL; i++)
                pAnycast = pAnycast->Next;
            printf("\tNumber of Anycast Addresses: %d\n", i);
        } else
            printf("\tNo Anycast Addresses\n");

        pMulticast = pCurrAddresses->FirstMulticastAddress;
        if (pMulticast) {
            for (i = 0; pMulticast != NULL; i++)
                pMulticast = pMulticast->Next;
            printf("\tNumber of Multicast Addresses: %d\n", i);
        } else
            printf("\tNo Multicast Addresses\n");
    }
}

```

```

        printf("\tNo Multicast Addresses\n");

pDnServer = pCurrAddresses->FirstDnsServerAddress;
if (pDnServer) {
    for (i = 0; pDnServer != NULL; i++)
        pDnServer = pDnServer->Next;
    printf("\tNumber of DNS Server Addresses: %d\n", i);
} else
    printf("\tNo DNS Server Addresses\n");

printf("\tDNS Suffix: %wS\n", pCurrAddresses->DnsSuffix);
printf("\tDescription: %wS\n", pCurrAddresses->Description);
printf("\tFriendly name: %wS\n", pCurrAddresses->FriendlyName);

if (pCurrAddresses->PhysicalAddressLength != 0) {
    printf("\tPhysical address: ");
    for (i = 0; i < pCurrAddresses->PhysicalAddressLength;
         i++) {
        if (i == (pCurrAddresses->PhysicalAddressLength - 1))
            printf("%.2X\n",
                   (int) pCurrAddresses->PhysicalAddress[i]);
        else
            printf("%.2X-", 
                   (int) pCurrAddresses->PhysicalAddress[i]);
    }
}
printf("\tFlags: %ld\n", pCurrAddresses->Flags);
printf("\tMtu: %lu\n", pCurrAddresses->Mtu);
printf("\tIfType: %ld\n", pCurrAddresses->IfType);
printf("\tOperStatus: %ld\n", pCurrAddresses->OperStatus);
printf("\tIpv6IfIndex (IPv6 interface): %u\n",
       pCurrAddresses->Ipv6IfIndex);
printf("\tZoneIndices (hex): ");
for (i = 0; i < 16; i++)
    printf("%lx ", pCurrAddresses->ZoneIndices[i]);
printf("\n");

pPrefix = pCurrAddresses->FirstPrefix;
if (pPrefix) {
    for (i = 0; pPrefix != NULL; i++)
        pPrefix = pPrefix->Next;
    printf("\tNumber of IP Adapter Prefix entries: %d\n", i);
} else
    printf("\tNo IP Adapter Prefix entries\n");

printf("\n");

pCurrAddresses = pCurrAddresses->Next;
}
} else {
    printf("Call to GetAdaptersAddresses failed with error: %d\n",
           dwRetVal);
    if (dwRetVal == ERROR_NO_DATA)
        printf("\tNo addresses were found for the requested
parameters\n");
}

```

```

    else {

        if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS, NULL, dwRetVal,
MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
(LPTSTR) & lpMsgBuf, 0, NULL)) {
            printf("\tError: %s", lpMsgBuf);
            LocalFree(lpMsgBuf);
            FREE(pAddresses);
            exit(1);
        }
    }
    FREE(pAddresses);
    return 0;
}

```

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetAdaptersAddresses](#)

[IF_OPER_STATUS](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ANYCAST_ADDRESS](#)

[IP_ADAPTER_DNS_SERVER_ADDRESS](#)

[IP_ADAPTER_DNS_SUFFIX](#)

[IP_ADAPTER_GATEWAY_ADDRESS](#)

[IP_ADAPTER_MULTICAST_ADDRESS](#)

[IP_ADAPTER_PREFIX](#)

[IP_ADAPTER_UNICAST_ADDRESS](#)

[IP_ADAPTER_WINS_SERVER_ADDRESS](#)

[SCOPE_LEVEL](#)

[SOCKET_ADDRESS](#)

[sockaddr](#)

反馈

此页面是否有帮助？

[!\[\]\(99c5664f767d68b2e1153a5b7e51fbd0_img.jpg\) 是](#)

[!\[\]\(81dfd8af6d67e25d84a4c0a0f860cc02_img.jpg\) 否](#)

[在 Microsoft Q&A 获得帮助](#)

IP_ADAPTER_ANYCAST_ADDRESS_XP 结构 (iptypes.h)

项目2023/08/24

IP_ADAPTER_ANYCAST_ADDRESS结构将单个任播 IP 地址存储在特定适配器的地址链接列表中。

语法

C++

```
typedef struct _IP_ADAPTER_ANYCAST_ADDRESS_XP {
    union {
        ULLONG Alignment;
        struct {
            ULONG Length;
            DWORD Flags;
        };
    };
    struct _IP_ADAPTER_ANYCAST_ADDRESS_XP *Next;
    SOCKET_ADDRESS Address;
} IP_ADAPTER_ANYCAST_ADDRESS_XP, *PIP_ADAPTER_ANYCAST_ADDRESS_XP;
```

成员

Alignment

类型: **ULLONG**

保留。由编译器用于对齐结构。

Length

类型: **ULONG**

此结构的长度（以字节为单位）。

Flags

类型: **DWORD**

此任播 IP 地址的标志。

下表列出了可能的值。这些常量在 *Iptypes.h* 头文件中定义。

值	含义
IP_ADAPTER_ADDRESS_DNS_ELIGIBLE	IP 地址在 DNS 中显示是合法的。
IP_ADAPTER_ADDRESS_TRANSIENT	IP 地址是群集地址，不应由大多数应用程序使用。

Next

类型: `struct _IP_ADAPTER_ANYCAST_ADDRESS*`

指向列表中下一个 anycast IP 地址结构的指针。

Address

类型: `SOCKET_ADDRESS`

此任播 IP 地址条目的 IP 地址。此成员可以是 IPv6 地址或 IPv4 地址。

注解

`IP_ADAPTER_ADDRESSES` 结构由 `GetAdaptersAddresses` 函数检索。

`IP_ADAPTER_ADDRESSES` 结构的 `FirstAnycastAddress` 成员是指向 `IP_ADAPTER_ANYCAST_ADDRESS` 结构链接列表的指针。

`SOCKET_ADDRESS` 结构用于 `IP_ADAPTER_ANYCAST_ADDRESS` 结构。在针对 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，`SOCKET_ADDRESS` 结构在 *Ws2def.h* 头文件中定义，*Wsock2.h* 头文件自动包含该头文件。在针对 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上，`SOCKET_ADDRESS` 结构在 *Winsock2.h* 头文件中声明。若要使用 `IP_ADAPTER_ANYCAST_ADDRESS` 结构，必须在 *Iphlpapi.h* 头文件之前包含 *Winsock2.h* 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	<i>Iptypes.h</i> (包括 <i>Iphlpapi.h</i>)

另请参阅

[GetAdaptersAddresses](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

[SOCKET_ADDRESS](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP_ADAPTER_DNS_SERVER_ADDRESS_XP 结构 (iptypes.h)

项目2023/08/24

IP_ADAPTER_DNS_SERVER_ADDRESS结构将单个 DNS 服务器地址存储在特定适配器的 DNS 服务器地址链接列表中。

语法

C++

```
typedef struct _IP_ADAPTER_DNS_SERVER_ADDRESS_XP {
    union {
        ULONGLONG Alignment;
        struct {
            ULONG Length;
            DWORD Reserved;
        };
    };
    struct _IP_ADAPTER_DNS_SERVER_ADDRESS_XP *Next;
    SOCKET_ADDRESS Address;
} IP_ADAPTER_DNS_SERVER_ADDRESS_XP, *PIP_ADAPTER_DNS_SERVER_ADDRESS_XP;
```

成员

Alignment

保留。由编译器用于对齐结构。

Length

此结构的长度（以字节为单位）。

Reserved

保留。

Next

指向列表中下一个 DNS 服务器地址结构的指针。

Address

此 DNS 服务器条目的 IP 地址。 此成员可以是 IPv6 地址或 IPv4 地址。

注解

[IP_ADAPTER_ADDRESSES](#) 结构由 [GetAdaptersAddresses](#) 函数检索。

[IP_ADAPTER_ADDRESSES](#) 结构的 [FirstDnsServerAddress](#) 成员是指向 [IP_ADAPTER_DNS_SERVER_ADDRESS](#) 结构链接列表的指针。

[SOCKET_ADDRESS](#) 结构用于 [IP_ADAPTER_DNS_SERVER_ADDRESS](#) 结构。 在针对 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，[SOCKET_ADDRESS](#) 结构在 *Ws2def.h* 头文件中定义，*Wsock2.h* 头文件自动包含该头文件。 在针对 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上，[SOCKET_ADDRESS](#) 结构在 *Winsock2.h* 头文件中声明。 若要使用 [IP_ADAPTER_DNS_SERVER_ADDRESS](#) 结构，必须在 *Iphlpapi.h* 头文件之前包含 *Winsock2.h* 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetAdaptersAddresses](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

[SOCKET_ADDRESS](#)

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获得帮助

IP_ADAPTER_DNS_SUFFIX 结构 (iptypes.h)

项目2023/08/24

IP_ADAPTER_DNS_SUFFIX结构将 DNS 后缀存储在特定适配器的 DNS 后缀链接列表中。

语法

C++

```
typedef struct _IP_ADAPTER_DNS_SUFFIX {
    struct _IP_ADAPTER_DNS_SUFFIX *Next;
    WCHAR String[MAX_DNS_SUFFIX_STRING_LENGTH];
} IP_ADAPTER_DNS_SUFFIX, *PIP_ADAPTER_DNS_SUFFIX;
```

成员

Next

指向链接列表中下一个 DNS 后缀的指针。

String[MAX_DNS_SUFFIX_STRING_LENGTH]

此 DNS 后缀条目的 DNS 后缀。

注解

IP_ADAPTER_ADDRESSES结构由 [GetAdaptersAddresses](#) 函数检索。

IP_ADAPTER_ADDRESSES 结构的 FirstDnsSuffix 成员是指向IP_ADAPTER_DNS_SUFFIX 结构链接列表的指针。

要求

最低受支持的客户端	Windows Vista SP1 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetAdaptersAddresses](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP_ADAPTER_GATEWAY_ADDRESS_LH 结构 (iptypes.h)

项目2023/08/24

IP_ADAPTER_GATEWAY_ADDRESS结构将单个网关地址存储在特定适配器的网关地址链接列表中。

语法

C++

```
typedef struct _IP_ADAPTER_GATEWAY_ADDRESS_LH {
    union {
        ULONGLONG Alignment;
        struct {
            ULONG Length;
            DWORD Reserved;
        };
    };
    struct _IP_ADAPTER_GATEWAY_ADDRESS_LH *Next;
    SOCKET_ADDRESS Address;
} IP_ADAPTER_GATEWAY_ADDRESS_LH, *PIP_ADAPTER_GATEWAY_ADDRESS_LH;
```

成员

Alignment

保留。由编译器用于对齐结构。

Length

此结构的长度（以字节为单位）。

Reserved

保留。

Next

指向列表中下一个网关地址结构的指针。

Address

此网关条目的 IP 地址。 此成员可以是 IPv6 地址或 IPv4 地址。

注解

[IP_ADAPTER_ADDRESSES](#) 结构由 [GetAdaptersAddresses](#) 函数检索。

[IP_ADAPTER_ADDRESSES](#) 结构的 [FirstGatewayAddress](#) 成员是指向 [IP_ADAPTER_GATEWAY_ADDRESS](#) 结构链接列表的指针。

[SOCKET_ADDRESS](#) 结构用于 [IP_ADAPTER_GATEWAY_ADDRESS](#) 结构。 在针对 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，[SOCKET_ADDRESS](#) 结构在 *Ws2def.h* 头文件中定义，*Wsock2.h* 头文件自动包含该头文件。 在针对 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上，[SOCKET_ADDRESS](#) 结构在 *Winsock2.h* 头文件中声明。 若要使用 [IP_ADAPTER_GATEWAY_ADDRESS](#) 结构，必须在 *Iphlpapi.h* 头文件之前包含 *Winsock2.h* 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetAdaptersAddresses](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

[SOCKET_ADDRESS](#)

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获得帮助

IP_ADAPTER_INDEX_MAP 结构 (ipexport.h)

项目2023/08/24

IP_ADAPTER_INDEX_MAP结构存储与启用了 IPv4 的网络适配器关联的接口索引以及网络适配器的名称。

语法

C++

```
typedef struct _IP_ADAPTER_INDEX_MAP {
    ULONG Index;
    WCHAR Name[MAX_ADAPTER_NAME];
} IP_ADAPTER_INDEX_MAP, *PIP_ADAPTER_INDEX_MAP;
```

成员

Index

与网络适配器关联的接口索引。

Name[MAX_ADAPTER_NAME]

指向包含适配器名称的 Unicode 字符串的指针。

注解

IP_ADAPTER_INDEX_MAP结构特定于启用了 IPv4 的网络适配器。

适配器索引可能会在禁用然后启用适配器时发生更改，或者在其他情况下，不应被视为永久性。

在 Windows Vista 及更高版本上，IP_ADAPTER_INDEX_MAP 结构的 Name 成员可能是网络接口的 GUID 的 Unicode 字符串，(字符串以“{”字符) 开头。

此结构在 *Ipexport.h* 头文件中定义，该文件自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Ipexport.h* 头文件。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[IP 帮助程序结构](#)

[IP 帮助程序起始页](#)

[IP_INTERFACE_INFO](#)

[IpReleaseAddress](#)

[IpRenewAddress](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP_ADAPTER_INFO 结构 (iptypes.h)

项目2023/08/24

IP_ADAPTER_INFO结构包含有关本地计算机上的特定网络适配器的信息。

语法

C++

```
typedef struct _IP_ADAPTER_INFO {
    struct _IP_ADAPTER_INFO *Next;
    DWORD ComboIndex;
    char AdapterName[MAX_ADAPTER_NAME_LENGTH + 4];
    char Description[MAX_ADAPTER_DESCRIPTION_LENGTH + 4];
    UINT AddressLength;
    BYTE Address[MAX_ADAPTER_ADDRESS_LENGTH];
    DWORD Index;
    UINT Type;
    UINT DhcpEnabled;
    PIP_ADDR_STRING CurrentIpAddress;
    IP_ADDR_STRING IpAddressList;
    IP_ADDR_STRING GatewayList;
    IP_ADDR_STRING DhcpServer;
    BOOL HaveWins;
    IP_ADDR_STRING PrimaryWinsServer;
    IP_ADDR_STRING SecondaryWinsServer;
    time_t LeaseObtained;
    time_t LeaseExpires;
} IP_ADAPTER_INFO, *PIP_ADAPTER_INFO;
```

成员

Next

类型: `struct _IP_ADAPTER_INFO*`

指向适配器列表中的下一个适配器的指针。

ComboIndex

类型: `DWORD`

保留。

`AdapterName[MAX_ADAPTER_NAME_LENGTH + 4]`

类型: `char[MAX_ADAPTER_NAME_LENGTH + 4]`

适配器名称的 ANSI 字符串。

`Description[MAX_ADAPTER_DESCRIPTION_LENGTH + 4]`

类型: `char[MAX_ADAPTER_DESCRIPTION_LENGTH + 4]`

包含适配器说明的 ANSI 字符串。

`AddressLength`

类型: `UINT`

适配器的硬件地址的长度 (以字节为单位)。

`Address[MAX_ADAPTER_ADDRESS_LENGTH]`

类型: `BYTE[MAX_ADAPTER_ADDRESS_LENGTH]`

表示为 `BYTE` 数组的适配器的硬件地址。

`Index`

类型: `DWORD`

适配器索引。

适配器索引可能会在禁用然后启用适配器时发生更改，或者在其他情况下，不应被视为永久性。

`Type`

类型: `UINT`

适配器类型。适配器类型的可能值列在 *Ipifcons.h* 头文件中。

下表列出了适配器类型的常见值，尽管其他值可以在 Windows Vista 及更高版本上使用。

值	含义
<code>MIB_IF_TYPE_OTHER</code> 1	一些其他类型的网络接口。
<code>MIB_IF_TYPE_ETHERNET</code> 6	以太网网络接口。
<code>IF_TYPE_ISO88025_TOKENRING</code>	<code>MIB_IF_TYPE_TOKENRING</code>

MIB_IF_TYPE PPP 23	PPP 网络接口。
MIB_IF_TYPE_LOOPBACK 24	软件环回网络接口。
MIB_IF_TYPE_SLIP 28	ATM 网络接口。
IF_TYPE_IEEE80211 71	IEEE 802.11 无线网络接口。

注意 此适配器类型在 Windows Vista 及更高版本上返回。在 Windows Server 2003 和 Windows XP 上，IEEE 802.11 无线网络接口返回 **适配器类型 MIB_IF_TYPE_ETHERNET**。

DhcpEnabled

类型: **UINT**

一个选项值，该值指定是否为此适配器启用动态主机配置协议 (DHCP)。

CurrentIpAddress

类型: **PIP_ADDR_STRING**

保留。

IpAddressList

类型: **IP_ADDR_STRING**

与此适配器关联的 IPv4 地址列表表示为 **IP_ADDR_STRING** 结构的链接列表。一个适配器可以分配多个 IPv4 地址。

GatewayList

类型: **IP_ADDR_STRING**

此适配器的网关的 IPv4 地址表示为 **IP_ADDR_STRING** 结构的链接列表。适配器可以分配多个 IPv4 网关地址。此列表通常包含此适配器的默认网关的 IPv4 地址的单个条目。

DhcpServer

类型: IP_ADDR_STRING

此适配器的 DHCP 服务器的 IPv4 地址表示为 IP_ADDR_STRING 结构的链接列表。此列表包含此适配器 DHCP 服务器的 IPv4 地址的单个条目。值为 255.255.255.255 表示无法访问 DHCP 服务器，或者正在达到 DHCP 服务器。

仅当 DhcpEnabled 成员为非零时，此成员才有效。

HaveWins

类型: BOOL

一个选项值，指定此适配器是否使用 Windows Internet 名称服务 (WINS)。

PrimaryWinsServer

类型: IP_ADDR_STRING

主 WINS 服务器的 IPv4 地址表示为 IP_ADDR_STRING 结构的链接列表。此列表包含此适配器的主 WINS 服务器 IPv4 地址的单个条目。

仅当 HaveWins 成员为 TRUE 时，此成员才有效。

SecondaryWinsServer

类型: IP_ADDR_STRING

辅助 WINS 服务器的 IPv4 地址表示为 IP_ADDR_STRING 结构的链接列表。适配器可以分配多个辅助 WINS 服务器地址。

仅当 HaveWins 成员为 TRUE 时，此成员才有效。

LeaseObtained

类型: time_t

获取当前 DHCP 租约的时间。

仅当 DhcpEnabled 成员为非零时，此成员才有效。

LeaseExpires

类型: time_t

当前 DHCP 租约到期的时间。

仅当 DhcpEnabled 成员为非零时，此成员才有效。

注解

IP_ADAPTER_INFO 结构仅限于有关本地计算机上的特定网络适配器的 IPv4 信息。通过调用 [GetAdaptersInfo](#) 函数检索 IP_ADAPTER_INFO 结构。

使用 Visual Studio 2005 及更高版本时，time_t 数据类型默认为 8 字节数据类型，而不是用于 32 位平台上 LeaseObtained 和 LeaseExpires 成员的 4 字节数据类型。若要在 32 位平台上正确使用 IP_ADAPTER_INFO 结构，请定义 _USE_32BIT_TIME_T (用作 -D _USE_32BIT_TIME_T 选项，例如，在编译应用程序以强制 time_t 数据类型为 4 字节数据类型时)。

为了在 Windows XP 及更高版本上使用，[IP_ADAPTER_ADDRESSES](#) 结构同时包含 IPv4 和 IPv6 信息。[GetAdaptersAddresses](#) 函数检索 IPv4 和 IPv6 适配器信息。

示例

此示例检索适配器信息并打印每个适配器的各种属性。

C++

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>
#pragma comment(lib, "IPHLPAPI.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))

/* Note: could also use malloc() and free() */

int __cdecl main()
{
    /* Declare and initialize variables */

    // It is possible for an adapter to have multiple
    // IPv4 addresses, gateways, and secondary WINS servers
    // assigned to the adapter.
    //
    // Note that this sample code only prints out the
    // first entry for the IP address/mask, and gateway, and
    // the primary and secondary WINS server for each adapter.

    PIP_ADAPTER_INFO pAdapterInfo;
    PIP_ADAPTER_INFO pAdapter = NULL;
    DWORD dwRetVal = 0;
    UINT i;
```

```

/* variables used to print DHCP time info */
struct tm newtime;
char buffer[32];
errno_t error;

ULONG ulOutBufLen = sizeof (IP_ADAPTER_INFO);
pAdapterInfo = (IP_ADAPTER_INFO *) MALLOC(sizeof (IP_ADAPTER_INFO));
if (pAdapterInfo == NULL) {
    printf("Error allocating memory needed to call GetAdaptersinfo\n");
    return 1;
}
// Make an initial call to GetAdaptersInfo to get
// the necessary size into the ulOutBufLen variable
if (GetAdaptersInfo(pAdapterInfo, &ulOutBufLen) ==
ERROR_BUFFER_OVERFLOW) {
    FREE(pAdapterInfo);
    pAdapterInfo = (IP_ADAPTER_INFO *) MALLOC(ulOutBufLen);
    if (pAdapterInfo == NULL) {
        printf("Error allocating memory needed to call
GetAdaptersinfo\n");
        return 1;
    }
}

if ((dwRetVal = GetAdaptersInfo(pAdapterInfo, &ulOutBufLen)) ==
NO_ERROR) {
    pAdapter = pAdapterInfo;
    while (pAdapter) {
        printf("\tComboIndex: \t%d\n", pAdapter->ComboIndex);
        printf("\tAdapter Name: \t%s\n", pAdapter->AdapterName);
        printf("\tAdapter Desc: \t%s\n", pAdapter->Description);
        printf("\tAdapter Addr: \t");
        for (i = 0; i < pAdapter->AddressLength; i++) {
            if (i == (pAdapter->AddressLength - 1))
                printf("%.2X\n", (int) pAdapter->Address[i]);
            else
                printf("%.2X-", (int) pAdapter->Address[i]);
        }
        printf("\tIndex: \t%d\n", pAdapter->Index);
        printf("\tType: \t");
        switch (pAdapter->Type) {
        case MIB_IF_TYPE_OTHER:
            printf("Other\n");
            break;
        case MIB_IF_TYPE_ETHERNET:
            printf("Ethernet\n");
            break;
        case MIB_IF_TYPE_TOKENRING:
            printf("Token Ring\n");
            break;
        case MIB_IF_TYPE_FDDI:
            printf("FDDI\n");
            break;
        case MIB_IF_TYPE_PPP:
            printf("PPP\n");
            break;
        }
    }
}

```

```

        break;
    case MIB_IF_TYPE_LOOPBACK:
        printf("Lookback\n");
        break;
    case MIB_IF_TYPE_SLIP:
        printf("Slip\n");
        break;
    default:
        printf("Unknown type %ld\n", pAdapter->Type);
        break;
    }

    printf("\tIP Address: \t%s\n",
           pAdapter->IpAddressListIpAddress.String);
    printf("\tIP Mask: \t%s\n", pAdapter-
>IpAddressListIpMask.String);

    printf("\tGateway: \t%s\n", pAdapter-
>GatewayListIpAddress.String);
    printf("\t***\n");

    if (pAdapter->DhcpEnabled) {
        printf("\tDHCP Enabled: Yes\n");
        printf("\t DHCP Server: \t%s\n",
               pAdapter->DhcpServerIpAddress.String);

        printf("\t Lease Obtained: ");
        /* Display local time */
        error = _localtime32_s(&newtime, (_time32_t*) &pAdapter-
>LeaseObtained);
        if (error)
            printf("Invalid Argument to _localtime32_s\n");
        else {
            // Convert to an ASCII representation
            error = asctime_s(buffer, 32, &newtime);
            if (error)
                printf("Invalid Argument to asctime_s\n");
            else
                /* asctime_s returns the string terminated by \n\0
*/
                printf("%s", buffer);
        }

        printf("\t Lease Expires: ");
        error = _localtime32_s(&newtime, (_time32_t*) &pAdapter-
>LeaseExpires);
        if (error)
            printf("Invalid Argument to _localtime32_s\n");
        else {
            // Convert to an ASCII representation
            error = asctime_s(buffer, 32, &newtime);
            if (error)
                printf("Invalid Argument to asctime_s\n");
            else
                /* asctime_s returns the string terminated by \n\0
*/

```

```

*/
    printf("%s", buffer);
}
} else
    printf("\tDHCP Enabled: No\n");

if (pAdapter->HaveWins) {
    printf("\tHave Wins: Yes\n");
    printf("\t Primary Wins Server: %s\n",
        pAdapter->PrimaryWinsServer.IpAddress.String);
    printf("\t Secondary Wins Server: %s\n",
        pAdapter->SecondaryWinsServer.IpAddress.String);
} else
    printf("\tHave Wins: No\n");
pAdapter = pAdapter->Next;
printf("\n");
}
} else {
    printf("GetAdaptersInfo failed with error: %d\n", dwRetVal);

}
if (pAdapterInfo)
    FREE(pAdapterInfo);

return 0;
}

```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetAdaptersAddresses](#)

[GetAdaptersInfo](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

IP_ADDRESS_STRING

IP_ADDR_STRING

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP_ADAPTER_MULTICAST_ADDRESS_XP 结构 (iptypes.h)

项目2023/08/24

IP_ADAPTER_MULTICAST_ADDRESS结构将单个多播地址存储在特定适配器的地址链接列表中。

语法

C++

```
typedef struct _IP_ADAPTER_MULTICAST_ADDRESS_XP {
    union {
        ULONGLONG Alignment;
        struct {
            ULONG Length;
            DWORD Flags;
        };
    };
    struct _IP_ADAPTER_MULTICAST_ADDRESS_XP *Next;
    SOCKET_ADDRESS Address;
} IP_ADAPTER_MULTICAST_ADDRESS_XP, *PIP_ADAPTER_MULTICAST_ADDRESS_XP;
```

成员

Alignment

类型: **ULLONG**

保留。由编译器用来对齐结构。

Length

类型: **ULONG**

此结构的长度（以字节为单位）。

Flags

类型: **DWORD**

此多播 IP 地址的一组标志。

下表列出了可能的值。这些常量在 *Iptypes.h* 头文件中定义。

值	含义
IP_ADAPTER_ADDRESS_DNS_ELIGIBLE	IP 地址在 DNS 中显示是合法的。
IP_ADAPTER_ADDRESS_TRANSIENT	IP 地址是群集地址，不应由大多数应用程序使用。

Next

类型: `struct _IP_ADAPTER_MULTICAST_ADDRESS*`

指向列表中下一个多播 IP 地址结构的指针。

Address

类型: `SOCKET_ADDRESS`

此多播 IP 地址条目的 IP 地址。此成员可以是 IPv6 地址或 IPv4 地址。

注解

`IP_ADAPTER_ADDRESSES` 结构由 `GetAdaptersAddresses` 函数检索。

`IP_ADAPTER_ADDRESSES` 结构的 `FirstMulticastAddress` 成员是指向 `IP_ADAPTER_MULTICAST_ADDRESS` 结构链接列表的指针。

`SOCKET_ADDRESS` 结构用于 `IP_ADAPTER_MULTICAST_ADDRESS` 结构。在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 中，头文件的组织已更改，`SOCKET_ADDRESS` 结构在 *Winsock2.h* 头文件自动包含的 *Ws2def.h* 头文件中定义。在为 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上，`SOCKET_ADDRESS` 结构在 *Winsock2.h* 头文件中声明。若要使用 `IP_ADAPTER_MULTICAST_ADDRESS` 结构，必须在 *Iphlpapi.h* 头文件之前包含 *Winsock2.h* 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	<i>Iptypes.h</i> (包括 <i>Iphlpapi.h</i>)

另请参阅

[GetAdaptersAddresses](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

[SOCKET_ADDRESS](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP_ADAPTER_ORDER_MAP 结构 (ipexport.h)

项目2023/08/24

IP_ADAPTER_ORDER_MAP结构在本地计算机上存储有关适配器及其相对优先级的信息数组。

语法

C++

```
typedef struct _IP_ADAPTER_ORDER_MAP {
    ULONG NumAdapters;
    ULONG AdapterOrder[1];
} IP_ADAPTER_ORDER_MAP, *PIP_ADAPTER_ORDER_MAP;
```

成员

NumAdapters

AdapterOrder 成员中的网络适配器数。

AdapterOrder[1]

本地计算机上的适配器索引数组，按**网络连接的“适配器和绑定”对话框中指定的顺序提供**。

注解

此结构由 GetAdapterOrderMap 函数返回，在网络操作期间用作其他相等接口的绑定断路器。不应直接调用 GetAdapterOrderMap 函数；请改用 GetAdaptersInfo 函数。

即使 IP_ADAPTER_ORDER_MAP 结构的 NumAdapters 成员指示没有网络适配器，IP_ADAPTER_ORDER_MAP 结构至少包含一个 AdapterOrder 成员。当 IP_ADAPTER_ORDER_MAP 结构的 NumAdapters 成员为零时，单个 AdapterOrder 成员的值是未定义的。

此结构在 *Ipexport.h* 头文件中定义，该文件自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Ipexport.h* 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[GetAdapterOrderMap](#)

[GetAdaptersInfo](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

(iptypes.h) IP_ADAPTER_PREFIX_XP 结构

项目2023/08/24

IP_ADAPTER_PREFIX结构存储 IP 地址前缀。

语法

C++

```
typedef struct _IP_ADAPTER_PREFIX_XP {
    union {
        ULONGLONG Alignment;
        struct {
            ULONG Length;
            DWORD Flags;
        };
    };
    struct _IP_ADAPTER_PREFIX_XP *Next;
    SOCKET_ADDRESS Address;
    ULONG PrefixLength;
} IP_ADAPTER_PREFIX_XP, *PIP_ADAPTER_PREFIX_XP;
```

成员

Alignment

保留。由编译器用于对齐结构。

Length

此结构的长度（以字节为单位）。

Flags

此成员是保留成员，应设置为零。

Next

指向列表中下一个适配器前缀结构的指针。

Address

地址前缀，采用 SOCKET_ADDRESS 结构的形式。

PrefixLength

前缀的长度 (以位为单位)。

注解

[IP_ADAPTER_ADDRESSES](#) 结构由 [GetAdaptersAddresses](#) 函数检索。在 Service Pack 1 (SP1) 及更高版本的 Windows XP 上, [IP_ADAPTER_ADDRESSES](#) 结构的 `FirstPrefix` 成员是指向[IP_ADAPTER_PREFIX](#)结构链接列表的指针。

[SOCKET_ADDRESS](#) 结构用于 [IP_ADAPTER_PREFIX](#) 结构。在针对 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上, 头文件的组织已更改, [SOCKET_ADDRESS](#) 结构在 `Ws2def.h` 头文件中定义, `Wsock2.h` 头文件自动包含该头文件。在针对 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上, [SOCKET_ADDRESS](#) 结构在 `Winsock2.h` 头文件中声明。若要使用 [IP_ADAPTER_PREFIX](#) 结构, 必须在 `Iphlpapi.h` 头文件之前包含 `Winsock2.h` 头文件。

要求

最低受支持的客户端	Windows Vista、带 SP1 的 Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	<code>iptypes.h</code> (包括 <code>Iphlpapi.h</code>)

另请参阅

[GetAdaptersAddresses](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

[SOCKET_ADDRESS](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

IP_ADAPTER_UNICAST_ADDRESS_LH结构 (iptypes.h)

项目2023/08/24

IP_ADAPTER_UNICAST_ADDRESS结构将单个单播 IP 地址存储在特定适配器的 IP 地址链接列表中。

语法

C++

```
typedef struct _IP_ADAPTER_UNICAST_ADDRESS_LH {
    union {
        ULONGLONG Alignment;
        struct {
            ULONG Length;
            DWORD Flags;
        };
    };
    struct _IP_ADAPTER_UNICAST_ADDRESS_LH *Next;
    SOCKET_ADDRESS Address;
    IP_PREFIX_ORIGIN PrefixOrigin;
    IP_SUFFIX_ORIGIN SuffixOrigin;
    IP_DAD_STATE DadState;
    ULONG ValidLifetime;
    ULONG PreferredLifetime;
    ULONG LeaseLifetime;
    UINT8 OnLinkPrefixLength;
} IP_ADAPTER_UNICAST_ADDRESS_LH, *PIP_ADAPTER_UNICAST_ADDRESS_LH;
```

成员

Alignment

Length

类型: ULONG

此结构的长度（以字节为单位）。

Flags

类型: DWORD

此 IP 地址的一组标志。

下表列出了可能的值。这些常量在 *Iptypes.h* 头文件中定义。

值	含义
IP_ADAPTER_ADDRESS_DNS_ELIGIBLE	IP 地址在 DNS 中显示是合法的。
IP_ADAPTER_ADDRESS_TRANSIENT	IP 地址是群集地址，不应由大多数应用程序使用。

[Next](#)

类型: `struct _IP_ADAPTER_UNICAST_ADDRESS*`

指向列表中下一个 IP 适配器地址结构的指针。

[Address](#)

类型: `SOCKET_ADDRESS`

此单播 IP 地址条目的 IP 地址。此成员可以是 IPv6 地址或 IPv4 地址。

[PrefixOrigin](#)

类型: `IP_PREFIX_ORIGIN`

IP 地址的前缀或网络部分。此成员可以是 *Iptypes.h* 头文件中定义的 `IP_PREFIX_ORIGIN` 枚举类型的值之一。

[SuffixOrigin](#)

类型: `IP_SUFFIX_ORIGIN`

IP 地址的后缀或主机部分。此成员可以是 *Iptypes.h* 头文件中定义的 `IP_SUFFIX_ORIGIN` 枚举类型的值之一。

[DadState](#)

类型: `IP_DAD_STATE`

重复地址检测 (DAD) 状态。此成员可以是 *Iptypes.h* 头文件中定义的 `IP_DAD_STATE` 枚举类型的值之一。重复地址检测适用于 IPv4 和 IPv6 地址。

[ValidLifetime](#)

类型: `ULONG`

IP 地址有效的最长生存期 (以秒为单位)。0xffffffff 值被视为无限。

`PreferredLifetime`

类型: **ULONG**

IP 地址有效的首选生存期 (以秒为单位)。0xffffffff 值被视为无限。

`LeaseLifetime`

类型: **ULONG**

IP 地址有效的租约生存期 (以秒为单位)。

`OnLinkPrefixLength`

类型: **UINT8**

IP 地址的前缀或网络部分的长度 (以位为单位)。对于单播 IPv4 地址, 任何大于 32 的值都是非法值。对于单播 IPv6 地址, 任何大于 128 的值都是非法值。值 255 通常用于表示非法值。

注意 此结构成员仅适用于 Windows Vista 及更高版本。

注解

`IP_ADAPTER_ADDRESSES` 结构由 [GetAdaptersAddresses](#) 函数检索。

`IP_ADAPTER_ADDRESSES` 结构的 `FirstUnicastAddress` 成员是指向 `IP_ADAPTER_UNICAST_ADDRESS` 结构链接列表的指针。

Windows Vista 及更高版本上 `IP_ADAPTER_UNICAST_ADDRESS` 结构的大小已更改。`Length` 成员应用于确定正在使用哪个版本的 `IP_ADAPTER_UNICAST_ADDRESS` 结构。

Windows Vista 和更高版本上的 `IP_ADAPTER_UNICAST_ADDRESS` 结构版本添加了以下新成员: `OnLinkPrefixLength`。

当此结构与 [GetAdaptersAddresses](#) 函数和类似的管理功能一起使用时, 将显示所有配置的地址, 包括重复的地址。静态配置地址时, 可能会发生此类重复的地址条目。此类报告有助于管理员进行故障排除。 `DadState` 成员可以有效地识别和排查此类情况。

在 Windows SDK 中, 在 Windows Vista 及更高版本上使用的结构版本定义为 `IP_ADAPTER_UNICAST_ADDRESS_LH`。在 Windows SDK 中, 此结构在早期系统 (包括 Windows XP service Pack 1 (SP1) 及更高版本) 上使用的版本定义为 `IP_ADAPTER_UNICAST_ADDRESS_XP`。如果目标平台是 Windows Vista 且更高版本

(`NTDDI_VERSION >= NTDDI_VISTA`、`_WIN32_WINNT >= 0x0600` 或 `WINVER >= 0x0600`)，则编译应用程序时，`IP_ADAPTER_UNICAST_ADDRESS_LH` 结构的类型为 `IP_ADAPTER_UNICAST_ADDRESS` 结构。如果目标平台不是 Windows Vista 及更高版本，则编译应用程序时，`IP_ADAPTER_UNICAST_ADDRESS_XP` 结构的类型为 `IP_ADAPTER_UNICAST_ADDRESS` 结构。

`SOCKET_ADDRESS` 结构用于 `IP_ADAPTER_UNICAST_ADDRESS` 结构。在针对 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，`SOCKET_ADDRESS` 结构在 `Ws2def.h` 头文件中定义，`Wsock2.h` 头文件自动包含该头文件。在针对 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上，`SOCKET_ADDRESS` 结构在 `Winsock2.h` 头文件中声明。若要使用 `IP_ADAPTER_UNICAST_ADDRESS` 结构，必须在 `Iphlpapi.h` 头文件之前包含 `Winsock2.h` 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	<code>iptypes.h</code> (包括 <code>Iphlpapi.h</code>)

另请参阅

[GetAdaptersAddresses](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

[IP_DAD_STATE](#)

[IP_PREFIX_ORIGIN](#)

[IP_SUFFIX_ORIGIN](#)

[SOCKET_ADDRESS](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP_ADAPTER_WINS_SERVER_ADDRESS_LH 结构 (iptypes.h)

项目2023/08/24

IP_ADAPTER_WINS_SERVER_ADDRESS结构将单个 Windows Internet 名称服务 (WINS) 服务器地址存储在特定适配器的 WINS 服务器地址链接列表中。

语法

C++

```
typedef struct _IP_ADAPTER_WINS_SERVER_ADDRESS_LH {
    union {
        ULLONG Alignment;
        struct {
            ULONG Length;
            DWORD Reserved;
        };
    };
    struct _IP_ADAPTER_WINS_SERVER_ADDRESS_LH *Next;
    SOCKET_ADDRESS Address;
} IP_ADAPTER_WINS_SERVER_ADDRESS_LH, *PIP_ADAPTER_WINS_SERVER_ADDRESS_LH;
```

成员

Alignment

保留。由编译器用来对齐结构。

Length

此结构的长度（以字节为单位）。

Reserved

保留。

Next

指向列表中下一个 WINS 服务器地址结构的指针。

Address

此 WINS 服务器条目的 IP 地址。 此成员可以是 IPv6 地址或 IPv4 地址。

注解

[IP_ADAPTER_ADDRESSES](#) 结构由 [GetAdaptersAddresses](#) 函数检索。

[IP_ADAPTER_ADDRESSES](#) 结构的 [FirstWinsServerAddress](#) 成员是指向 [IP_ADAPTER_WINS_SERVER_ADDRESS](#) 结构链接列表的指针。

[SOCKET_ADDRESS](#) 结构用于 [IP_ADAPTER_GATEWAY_ADDRESS](#) 结构。 在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 中，头文件的组织已更改，[SOCKET_ADDRESS](#) 结构在 *Winsock2.h* 头文件自动包含的 *Ws2def.h* 头文件中定义。 在为 Windows Server 2003 和 Windows XP 发布的平台软件开发工具包 (SDK) 上，[SOCKET_ADDRESS](#) 结构在 *Winsock2.h* 头文件中声明。 若要使用 [IP_ADAPTER_GATEWAY_ADDRESS](#) 结构，必须在 *Iphlpapi.h* 头文件之前包含 *Winsock2.h* 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetAdaptersAddresses](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_ADDRESSES](#)

[SOCKET_ADDRESS](#)

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获得帮助

IP_ADDR_STRING 结构 (iptypes.h)

项目2023/08/24

IP_ADDR_STRING结构表示 IPv4 地址链接列表中的节点。

语法

C++

```
typedef struct _IP_ADDR_STRING {
    struct _IP_ADDR_STRING *Next;
    IP_ADDRESS_STRING     IpAddress;
    IP_MASK_STRING        IpMask;
    DWORD                 Context;
} IP_ADDR_STRING, *PIP_ADDR_STRING;
```

成员

Next

指向列表中下一个IP_ADDR_STRING 结构的指针。

IpAddress

一个值，该值指定具有单个成员 String 的结构类型。 String 成员是大小为 16 的字符串数组。此数组保存一个 IPv4 地址（以点状十进制表示法表示）。

IpMask

一个值，该值指定具有单个成员 String 的结构类型。 String 成员是大小为 16 的字符串数组。此数组以点十进制表示法保存 IPv4 子网掩码。

Context

NTE) (网络表条目。此值对应于 [AddIPAddress](#) 和 [DeleteIPAddress](#) 函数中的 NTEContext 参数。

要求

最低受支持的客户端

Windows 2000 Professional [仅限桌面应用]

最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[AddIPAddress](#)

[DeleteIPAddress](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_INFO](#)

[IP_ADDRESS_STRING](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

(netioapi.h) IP_ADDRESS_PREFIX 结构

项目2023/08/25

IP_ADDRESS_PREFIX结构存储 IP 地址前缀。

语法

C++

```
typedef struct _IP_ADDRESS_PREFIX {
    SOCKADDR_INET Prefix;
    UINT8          PrefixLength;
} IP_ADDRESS_PREFIX, *PIP_ADDRESS_PREFIX;
```

成员

Prefix

IP 的前缀或网络部分，该地址表示为 IP 地址。

SOCKADDR_INET联合在 *Ws2ipdef.h* 标头中定义。

PrefixLength

IP 地址的前缀或网络部分的长度（以位为单位）。对于单播 IPv4 地址，任何大于 32 的值都是非法值。对于单播 IPv6 地址，任何大于 128 的值都是非法值。值 255 通常用于表示非法值。

注解

IP_ADDRESS_PREFIX结构在 Windows Vista 及更高版本上定义。

IP_ADDRESS_PREFIX 结构是 MIB_IPFORWARD_ROW2 结构中 DestinationPrefix 成员的数据类型。许多函数使用 MIB_IPFORWARD_ROW2 结构，包括 CreateIpForwardEntry2、DeleteIpForwardEntry2、GetBestRoute2、GetIpForwardEntry2、GetIpForwardTable2、InitializeIpForwardEntry、NotifyRouteChange2 和 SetIpForwardEntry2。

要求

最低受支持的客户端	无受支持的版本
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	netioapi.h (包括 Iphlpapi.h)

请参阅

[CreateIpForwardEntry2](#)

[DeleteIpForwardEntry2](#)

[GetBestRoute2](#)

[GetIpForwardEntry2](#)

[GetIpForwardTable2](#)

[InitializeIpForwardEntry](#)

[MIB_IPFORWARD_ROW2](#)

[NotifyRouteChange2](#)

[SOCKADDR_INET](#)

[SetIpForwardEntry2](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP_ADDRESS_STRING 结构 (iptypes.h)

项目2023/08/24

IP_ADDRESS_STRING结构以点十进制表示法存储 IPv4 地址。 IP_ADDRESS_STRING结构定义也是IP_MASK_STRING结构的类型定义。

语法

C++

```
typedef struct {
    char *String[4 4];
} IP_ADDRESS_STRING, *PIP_ADDRESS_STRING, IP_MASK_STRING, *PIP_MASK_STRING;
```

成员

String[4 * 4]

一个字符串，表示 IPv4 地址或 IPv4 子网掩码（以点十进制表示法表示）。

注解

IP_ADDRESS_STRING 结构用作[IP_ADDR_STRING](#)结构中的参数。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP_INTERFACE_INFO 结构 (ipexport.h)

项目2023/08/24

IP_INTERFACE_INFO结构包含本地系统上启用了 IPv4 的网络接口适配器的列表。

语法

C++

```
typedef struct _IP_INTERFACE_INFO {
    LONG             NumAdapters;
    IP_ADAPTER_INDEX_MAP Adapter[1];
} IP_INTERFACE_INFO, *PIP_INTERFACE_INFO;
```

成员

NumAdapters

适配器成员指向的数组中列出的 **适配器** 数。

Adapter[1]

[IP_ADAPTER_INDEX_MAP](#) 结构的数组。 每个结构将适配器索引映射到该适配器的名称。 适配器索引可能会在禁用然后启用适配器时发生更改，或者在其他情况下，不应被视为永久性。

注解

IP_INTERFACE_INFO结构特定于启用了 IPv4 的网络适配器。 IP_INTERFACE_INFO结构包含本地系统上启用了 IPv4 的网络适配器数，以及一组[IP_ADAPTER_INDEX_MAP](#)结构，其中包含每个启用了 IPv4 的网络适配器的信息。 即使 IP_INTERFACE_INFO 结构的 NumAdapters 成员指示未启用具有 IPv4 的网络适配器，IP_INTERFACE_INFO 结构也至少包含一个IP_ADAPTER_INDEX_MAP结构。 当 IP_INTERFACE_INFO 结构的 NumAdapters 成员为零时，IP_INTERFACE_INFO 结构中返回的单个 IP_ADAPTER_INDEX_MAP结构的成员的值未定义。

IP_INTERFACE_INFO 结构不能用于返回有关环回接口的信息。

在 Windows Vista 及更高版本中，IP_INTERFACE_INFO 结构中[IP_ADAPTER_INDEX_MAP](#) 结构的 Name 成员可能是网络接口 GUID 的 Unicode 字符串，(字符串以“{”字符) 开头。

此结构在 *Ipexport.h* 头文件中定义，该文件自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Ipexport.h* 头文件。

示例

以下示例检索本地系统上启用了 IPv4 的网络适配器列表，并打印第一个适配器的各种属性。

C++

```
// Declare and initialize variables
PIP_INTERFACE_INFO pInfo;
pInfo = (IP_INTERFACE_INFO *) malloc( sizeof(IP_INTERFACE_INFO) );
ULONG ulOutBufLen = 0;
DWORD dwRetVal = 0;

// Make an initial call to GetInterfaceInfo to get
// the necessary size in the ulOutBufLen variable
if ( GetInterfaceInfo(pInfo, &ulOutBufLen) == ERROR_INSUFFICIENT_BUFFER ) {
    free(pInfo);
    pInfo = (IP_INTERFACE_INFO *) malloc (ulOutBufLen);
}

// Make a second call to GetInterfaceInfo to get
// the actual data we need
if ((dwRetVal = GetInterfaceInfo(pInfo, &ulOutBufLen)) == NO_ERROR ) {
    printf("\tAdapter Name: %ws\n", pInfo->Adapter[0].Name);
    printf("\tAdapter Index: %ld\n", pInfo->Adapter[0].Index);
    printf("\tNum Adapters: %ld\n", pInfo->NumAdapters);

    // free memory allocated
    free(pInfo);
    pInfo = NULL;
}
else if (dwRetVal == ERROR_NO_DATA) {
    printf("There are no network adapters with IPv4 enabled on the local
system\n");
}

else {
    printf("GetInterfaceInfo failed.\n");
    LPVOID lpMsgBuf;

    if (FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        dwRetVal,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
        (LPTSTR) &lpMsgBuf,
```

```
    0,  
    NULL )) {  
    printf("\tError: %s", lpMsgBuf);  
}  
LocalFree( lpMsgBuf );  
}
```

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[GetInterfaceInfo](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADAPTER_INDEX_MAP](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP_INTERFACE_NAME_INFO_W2KSP1 结构 (iptypes.h)

项目2023/08/24

IP_INTERFACE_NAME_INFO 结构包含有关本地计算机上的 IPv4 接口的信息。

语法

C++

```
typedef struct ip_interface_name_info_w2ksp1 {
    ULONG Index;
    ULONG MediaType;
    UCHAR ConnectionType;
    UCHAR AccessType;
    GUID DeviceGuid;
    GUID InterfaceGuid;
} IP_INTERFACE_NAME_INFO_W2KSP1, *PIP_INTERFACE_NAME_INFO_W2KSP1;
```

成员

Index

类型: ULONG

活动实例的 IP 接口的索引。

MediaType

类型: ULONG

Internet 分配名称机构定义的接口类型 (IANA)。 Ipfcons.h 头文件中列出了接口类型的可能值。

下表列出了 接口类型的常见值：不过，还可以使用许多其他值。

值	含义
IF_TYPE_OTHER 1	一些其他类型的网络接口。
IF_TYPE_ETHERNET_CSMACD 6	以太网网络接口。

IF_TYPE_ISO88025_TOKENRING 9	令牌环网络接口。
IF_TYPE_PPP 23	PPP 网络接口。
IF_TYPE_SOFTWARE_LOOPBACK 24	软件环回网络接口。
IF_TYPE_ATM 37	ATM 网络接口。
IF_TYPE_IEEE80211 71	IEEE 802.11 无线网络接口。在 Windows Vista 及更高版本中，无线网卡报告为 IF_TYPE_IEEE80211。 Windows Server 2003、Windows 2000 Server SP1 和 Windows XP/2000：无线网卡报告为 IF_TYPE_ETHERNET_CSMACD。
IF_TYPE_TUNNEL 131	隧道类型封装网络接口。
IF_TYPE_IEEE1394 144	IEEE 1394 (Firewire) 高性能串行总线网络接口。

ConnectionType

类型: UCHAR

适配器的接口连接类型。

此成员的可能值在 Ipfcons.h 头文件中定义。

值	含义
IF_CONNECTION_DEDICATED 1	连接类型是专用的。当媒体感知为 TRUE 时，连接会自动启动。例如，以太网连接是专用的。
IF_CONNECTION_PASSIVE 2	连接类型为被动连接。远程端必须启动与本地工作站的连接。例如，RAS 接口是被动的。
IF_CONNECTION_DEMAND 3	连接类型为按需拨号。此类型的连接是为了响应本地操作(发送数据包，例如)。

AccessType

类型: UCHAR

IF_ACCESS_TYPE 枚举的值，该值指定接口的访问类型。

Windows Server 2003、Windows 2000 Server SP1 和 Windows XP/2000: 此成员的可能值在 `Ipfcons.h` 头文件中定义。

值	含义
<code>IF_ACCESS_LOOPBACK</code> 1	环回访问类型。此值指示接口以接收数据的形式循环回传输数据。
<code>IF_ACCESS_BROADCAST</code> 2	包括以太网的 LAN 访问类型。此值指示接口为多播或广播服务提供本机支持。
<code>IF_ACCESS_POINT_TO_POINT</code> 3	点到点访问类型。此值表示支持 CoNDIS/WAN，非广播多访问 (NBMA) 接口除外。 Windows Server 2003、Windows 2000 Server SP1 和 Windows XP/2000: 此值在 <code>Ipfcons.h</code> 头文件中定义为 <code>IF_ACCESS_POINTTOPOINT</code> 。
<code>IF_ACCESS_POINT_TO_MULTI_POINT</code> 4	点到多点访问类型。此值表示支持非广播多访问媒体，包括 RAS 内部接口和本机 ATM。 Windows Server 2003、Windows 2000 Server SP1 和 Windows XP/2000: 此值在 <code>Ipfcons.h</code> 头文件中定义为 <code>IF_ACCESS_POINTTOMULTIPOINT</code> 。

DeviceGuid

类型: GUID

标识接口的基础设备的 GUID。此成员可以是零 GUID。

InterfaceGuid

类型: GUID

标识映射到设备的接口的 GUID。可选。此成员可以是零 GUID。

注解

在 Microsoft Windows 软件开发工具包 (SDK) 中，用于 Windows 2000 Service Pack 1 (SP1) 及更高版本的结构版本定义为 `IP_INTERFACE_NAME_INFO_W2KSP1`。编译应用程序时，如果目标平台是 Windows 2000 SP1 和更高版本 (`NTDDI_VERSION >= NTDDI_WIN2KSP1`、`_WIN32_WINNT >= 0x0500` 或 `WINVER >= 0x0500`)，则

`IP_INTERFACE_NAME_INFO_W2KSP1` 结构的类型为 `IP_INTERFACE_NAME_INFO` 结构。

`MediaType`、`ConnectionType` 和 `AccessType` 成员、定义和分配的值可从 `Ipfcons.h` 头文件获取。

可选 InterfaceGuid 成员通常为拨号接口设置，可用于区分共享同一设备 GUID 的多个拨号接口。

要求

最低受支持的客户端	具有 SP1 的 Windows XP、Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows Server 2003、Windows 2000 Server SP1 [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetAdaptersAddresses](#)

[NhpAllocateAndGetInterfaceInfoFromStack](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

IP_MCAST_COUNTER_INFO 结构 (ipexport.h)

项目2023/08/24

IP_MCAST_COUNTER_INFO结构存储有关多播流量的统计信息。

语法

C++

```
typedef struct _IP_MCAST_COUNTER_INFO {
    ULONG64 InMcastOctets;
    ULONG64 OutMcastOctets;
    ULONG64 InMcastPkts;
    ULONG64 OutMcastPkts;
} IP_MCAST_COUNTER_INFO, *PIP_MCAST_COUNTER_INFO;
```

成员

InMcastOctets

收到的多播八进制数。

OutMcastOctets

传输的多播八进制数。

InMcastPkts

收到的多播数据包数。

OutMcastPkts

传输的多播数据包数。

注解

此结构在 *Ipexport.h* 头文件中定义，该文件自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Ipexport.h* 头文件。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP_OPTION_INFORMATION 结构 (ipexport.h)

项目2023/08/24

IP_OPTION_INFORMATION 结构描述了要包含在 IP 数据包标头中的选项。

语法

C++

```
typedef struct ip_option_information {
    UCHAR Ttl;
    UCHAR Tos;
    UCHAR Flags;
    UCHAR OptionsSize;
    PUCHAR OptionsData;
} IP_OPTION_INFORMATION, *PIP_OPTION_INFORMATION;
```

成员

Ttl

类型: UCHAR

IPv4 数据包标头中的生存时间字段。这是 IPv6 标头中的跃点限制字段。

Tos

类型: UCHAR

IPv4 标头中的服务字段的类型。当前以无提示方式忽略此成员。

Flags

类型: UCHAR

标志字段。在 IPv4 中，这是 IPv4 标头中的“标志”字段。在 IPv6 中，此字段由选项标头表示。

对于 IPv4，Flags 成员的可能值是 *Ipexport.h* 头文件中定义的以下值的组合：

值	含义
---	----

IP_FLAG_REVERSE 0x01	此值会导致 IP 数据包将添加到具有源的 IP 路由标头中。 此值仅适用于 Windows Vista 及更高版本。
IP_FLAG_DF 0x02	此值指示不应对数据包进行碎片处理。

OptionsSize

类型: UCHAR

IP 选项数据的大小 (以字节为单位)。

OptionsData

类型: PUCHAR

指向选项数据的指针。

注解

IP_OPTION_INFORMATION 结构用于描述要包含在 IP 数据包标头中的选项。 在 64 位平台上，应使用 [IP_OPTION_INFORMATION32](#) 结构。

TTL、TOS 和 Flags 成员中的值在 IP 标头中的特定字段中携带。

OptionsData 成员中的字节包含在标准 IP 标头后面的选项区域中。

除了 IPv4 的源路由选项外，选项数据必须采用 [RFC 791](#) 中指定的要在网络上传输的格式。IPv4 源路由选项应包含路由数据中的完整路由，即第一跃点到最终目标。从数据中提取第一个跃点，并相应地重新设置选项格式。否则，路由选项的格式应按照 [RFC 791](#) 中的指定。

若要与 IPv6 一起使用，选项数据的格式必须按照 [RFC 2460](#) 中指定的格式在线路上传输。

IP_OPTION_INFORMATION 结构是 `IcmpSendEcho`、`IcmpSendEcho2` 和 `Icmp6SendEcho2` 函数使用的 `ICMP_ECHO_REPLY` 结构的成员。

此结构在 `Ipexport.h` 头文件中定义，该文件自动包含在 `Iphlpapi.h` 头文件中。永远不应直接使用 `Ipexport.h` 头文件。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[ICMP_ECHO_REPLY](#)

[IP_OPTION_INFORMATION32](#)

[Icmp6Sendecho2](#)

[IcmpSendecho](#)

[IcmpSendecho2](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP_OPTION_INFORMATION32 结构 (ipexport.h)

项目2023/08/24

IP_OPTION_INFORMATION32 结构描述了要包含在 64 位平台上 IP 数据包标头中的选项。

语法

C++

```
typedef struct ip_option_information32 {
    UCHAR Ttl;
    UCHAR Tos;
    UCHAR Flags;
    UCHAR OptionsSize;
    UCHAR *OptionsData;
} IP_OPTION_INFORMATION32, *PIP_OPTION_INFORMATION32;
```

成员

Ttl

类型: UCHAR

IPv4 数据包标头中的生存时间字段。这是 IPv6 标头中的跃点限制字段。

Tos

类型: UCHAR

IPv4 标头中的服务字段的类型。当前以无提示方式忽略此成员。

Flags

类型: UCHAR

标志字段。在 IPv4 中，这是 IPv4 标头中的“标志”字段。在 IPv6 中，此字段由选项标头表示。

对于 IPv4，Flags 成员的可能值是 *Ipexport.h* 头文件中定义的以下值的组合：

值

含义

IP_FLAG_REVERSE 0x01	此值会导致 IP 数据包将添加到具有源的 IP 路由标头中。 此值仅适用于 Windows Vista 及更高版本。
IP_FLAG_DF 0x02	此值指示不应对数据包进行碎片处理。

OptionsSize

类型: UCHAR

IP 选项数据的大小 (以字节为单位)。

OptionsData

类型: UCHAR * POINTER_32

指向选项数据的指针。

注解

IP_OPTION_INFORMATION32 结构用于描述要包含在 64 位平台上 IP 数据包标头中的选项。仅当定义了 _WIN64 时，才会定义IP_OPTION_INFORMATION32结构。

TTL、TOS 和 Flags 成员中的值在 IP 标头中的特定字段中携带。

OptionsData 成员中的字节包含在标准 IP 标头后面的选项区域中。

除了 IPv4 的源路由选项外，选项数据必须采用 [RFC 791](#) 中指定的要在网络上传输的格式。IPv4 源路由选项应包含路由数据中的完整路由，即第一跃点到最终目标。从数据中提取第一个跃点，并相应地重新设置选项格式。否则，路由选项的格式应按照 [RFC 791](#) 中的指定。

若要与 IPv6 一起使用，选项数据的格式必须按照 [RFC 2460](#) 中指定的格式在线路上传输。

IP_OPTION_INFORMATION32 结构是 `IcmpSendEcho`、`IcmpSendEcho2` 和 `Icmp6SendEcho2` 函数使用的 `ICMP_ECHO_REPLY32` 结构的成员。

此结构在 `Ipexport.h` 头文件中定义，该文件自动包含在 `Iphlpapi.h` 头文件中。永远不应直接使用 `Ipexport.h` 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[ICMP_ECHO_REPLY32](#)

[IP_OPTION_INFORMATION](#)

[Icmp6Sendecho2](#)

[IcmpSendecho](#)

[IcmpSendecho2](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP_PER_ADAPTER_INFO_W2KSP1 结构 (iptypes.h)

项目2023/08/24

IP_PER_ADAPTER_INFO 结构包含特定于特定适配器的信息。

语法

C++

```
typedef struct _IP_PER_ADAPTER_INFO_W2KSP1 {
    UINT          AutoconfigEnabled;
    UINT          AutoconfigActive;
    PIP_ADDR_STRING CurrentDnsServer;
    IP_ADDR_STRING  DnsServerList;
} IP_PER_ADAPTER_INFO_W2KSP1, *PIP_PER_ADAPTER_INFO_W2KSP1;
```

成员

AutoconfigEnabled

指定是否在此适配器上启用 IP 地址自动配置 (APIPA)。请参阅“备注”。

AutoconfigActive

指定此适配器的 IP 地址当前是否由 APIPA 自动配置。

CurrentDnsServer

保留。使用 DnsServerList 成员获取本地计算机的 DNS 服务器。

DnsServerList

IP_ADDR_STRING 结构的链接列表，这些结构指定本地计算机使用的 DNS 服务器集。

注解

APIPA 使用 IANA 保留的 B 类网络 169.254.0.0（子网掩码为 255.255.0.0），在没有 DHCP 服务器的网络上启用自动 IP 地址配置。客户端发送 ARP 消息以确保所选地址当前未使用。以这种方式自动配置的客户端继续每五分钟轮询一次有效的 DHCP 服务器，如果找到，DHCP 服务器配置将覆盖所有自动配置设置。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	iptypes.h (包括 Iphlpapi.h)

另请参阅

[GetPerAdapterInfo](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_ADDR_STRING](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IP_UNIDIRECTIONAL_ADAPTER_ADDRESSES 结构 (ipexport.h)

项目2023/08/24

IP_UNIDIRECTIONAL_ADAPTER_ADDRESS结构存储与单向适配器关联的 IPv4 地址。

语法

C++

```
typedef struct _IP_UNIDIRECTIONAL_ADAPTER_ADDRESS {
    ULONG NumAdapters;
    IPAddr Address[1];
} IP_UNIDIRECTIONAL_ADAPTER_ADDRESS, *PIP_UNIDIRECTIONAL_ADAPTER_ADDRESS;
```

成员

NumAdapters

Address 成员指向的 IPv4 地址数。

Address[1]

IPAddr 类型的变量数组。 数组的每个元素指定与此单向适配器关联的 IPv4 地址。

注解

IP_UNIDIRECTIONAL_ADAPTER_ADDRESS 结构由 [GetUnidirectionalAdapterInfo](#) 函数检索。 单向适配器是可以接收 IPv4 数据报但无法传输它们的适配器。

要求

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	ipexport.h (包括 Iphlpapi.h)

另请参阅

[GetUnidirectionalAdapterInfo](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IPAddr](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IPV6_ADDRESS_EX 结构 (ipexport.h)

项目2023/08/24

IPV6_ADDRESS_EX结构存储 IPv6 地址。

语法

C++

```
typedef struct _IPV6_ADDRESS_EX {
    USHORT sin6_port;
    ULONG sin6_flowinfo;
    USHORT sin6_addr[8];
    ULONG sin6_scope_id;
} IPV6_ADDRESS_EX, *PIPV6_ADDRESS_EX;
```

成员

`sin6_port`

按网络字节顺序排列的 IPv6 端口号。

`sin6_flowinfo`

IPv6 标头中的 IPv6 flowinfo 值（按网络字节顺序排列）。

`sin6_addr[8]`

按网络字节顺序排列的 IPv6 地址。

`sin6_scope_id`

按网络字节顺序排列的 IPv6 范围 ID。

注解

IPV6_ADDRESS_EX 结构是 [Icmp6ParseReplies](#) 函数使用的 [ICMPV6_ECHO_REPLY](#) 结构的成员。

IPV6_ADDRESS_EX 结构在 Microsoft Windows 软件开发工具包 (SDK) 中包含的公共头文件中定义，但此结构由 Windows XP 和更高版本的 [Icmp6ParseReplies](#) 函数使用。

在 Windows SDK 中，如果目标平台是 Windows XP，则 `IPV6_ADDRESS_EX` 是编译应用程序时定义的结构，并且更高版本 (`NTDDI_VERSION >= NTDDI_XP`、`_WIN32_WINNT >= 0x0501` 或 `WINVER >= 0x0501`)。如果目标平台不是 Windows XP 及更高版本，则编译应用程序时，`IPV6_ADDRESS_EX` 结构未定义。

此结构在 `Ipxport.h` 头文件中定义，该文件自动包含在 `Iphlpapi.h` 头文件中。永远不应直接使用 `Ipxport.h` 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	<code>ipxport.h</code> (包括 <code>Iphlpapi.h</code>)

另请参阅

[ICMPV6_ECHO_REPLY](#)

[IP 帮助程序起始页](#)

[IP 帮助程序结构](#)

[IP_OPTION_INFORMATION](#)

[Icmp6CreateFile](#)

[Icmp6ParseReplies](#)

[Icmp6SendEcho2](#)

[IcmpCloseHandle](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

iphlpapi.h) (NET_ADDRESS_INFO 结构

项目2023/08/24

NET_ADDRESS_INFO 结构包含 [ParseNetworkString](#) 函数返回的 IP 地址信息。

语法

C++

```
typedef struct NET_ADDRESS_INFO_ {
    NET_ADDRESS_FORMAT Format;
    union {
        struct {
            WCHAR Address[DNS_MAX_NAME_BUFFER_LENGTH];
            WCHAR Port[6];
        } NamedAddress;
        SOCKADDR_IN Ipv4Address;
        SOCKADDR_IN6 Ipv6Address;
        SOCKADDR     IpAddress;
    };
} NET_ADDRESS_INFO, *PNET_ADDRESS_INFO;
```

成员

Format

类型: NET_ADDRESS_FORMAT

此结构中联合中的网络地址的格式。此成员是 *iphlpapi.h* 头文件中声明的 [NET_ADDRESS_FORMAT](#) 枚举中的枚举值。

NamedAddress

DNS 命名地址和端口。

NamedAddress.Address[DNS_MAX_NAME_BUFFER_LENGTH]

类型: WCHAR[DNS_MAX_NAME_BUFFER_LENGTH] 格式化为以 NULL 结尾的宽字符串的 DNS 名称。此字符串的最大长度是在 *Windns.h* 头文件中定义的 **DNS_MAX_NAME_BUFFER_LENGTH** 常量。

NamedAddress.Port[6]

类型: WCHAR[6] 以 NULL 结尾的宽字符串格式的网络端口。

`Ipv4Address`

类型: `SOCKADDR_IN`

表示为 `SOCKADDR_IN` 结构的 IPv4 地址。

`Ipv6Address`

类型: `SOCKADDR_IN6`

表示为 `SOCKADDR_IN6` 结构的 IPv6 地址。

`IpAddress`

类型: `SOCKADDR`

表示为 `SOCKADDR` 结构的 IPv4 或 IPv6 地址。

注解

`NET_ADDRESS_INFO` 结构在 Windows Vista 及更高版本上定义。

`NET_ADDRESS_INFO` 结构由 [ParseNetworkString](#) 函数返回。

`SOCKADDR_IN`、`SOCKADDR_IN6` 和 `SOCKADDR` 结构在 `NET_ADDRESS_INFO` 结构中使用。 `SOCKADDR_IN` 和 `SOCKADDR` 结构在 `Ws2def.h` 头文件中定义，`Winsock2.h` 头文件会自动包含该文件。`SOCKADDR_IN6` 结构在 `Ws2ipdef.h` 头文件中定义，`Ws2tcpip.h` 头文件会自动包含该文件。 若要使用 `NET_ADDRESS_INFO` 结构，必须在 `Iphlpapi.h` 头文件之前包括 `Winsock2.h` 和 `Ws2tcpip.h` 头文件。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	<code>iphlpapi.h</code>

另请参阅

[NET_ADDRESS_FORMAT](#)

[ParseNetworkString](#)

[SOCKADDR](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

NET_LUID_LH union (ifdef.h)

项目2023/08/24

NET_LUID联合是网络接口的本地唯一标识符 (LUID)。

语法

C++

```
typedef union _NET_LUID_LH {
    ULONG64 Value;
    struct {
        ULONG64 Reserved : 24;
        ULONG64 NetLuidIndex : 24;
        ULONG64 IfType : 16;
    } Info;
} NET_LUID_LH, *PNET_LUID_LH;
```

成员

Value

类型: ULONG64

表示 LUID 的 64 位值。

Info

包含 64 位 LUID Value 成员中的组件字段的命名联合。

Info.Reserved

类型: ULONG64 此字段为保留字段。

Info.NetLuidIndex

类型: ULONG64 网络接口 LUID 索引。

Info.IfType

类型: ULONG64 Internet 分配名称机构 (IANA) 定义的接口类型。 *Ipifcons.h* 包含文件中列出了接口类型的可能值。

下表列出了接口类型的常见值，尽管可能还有许多其他值。

值	含义
IF_TYPE_OTHER 1	其他一些类型的网络接口。
IF_TYPE_ETHERNET_CSMACD 6	以太网网络接口。
IF_TYPE_ISO88025_TOKENRING 9	令牌环网络接口。
IF_TYPE_PPP 23	PPP 网络接口。
IF_TYPE_SOFTWARE_LOOPBACK 24	软件环回网络接口。
IF_TYPE_ATM 37	ATM 网络接口。
IF_TYPE_IEEE80211 71	IEEE 802.11 无线网络接口。
IF_TYPE_TUNNEL 131	隧道类型封装网络接口。
IF_TYPE_IEEE1394 144	IEEE 1394 (Firewire) 高性能串行总线网络接口。

注解

NET_LUID结构与协议无关，适用于 IPv6 和 IPv4 协议的网络接口。**NET_LUID**结构在 Windows Vista 及更高版本上定义。

IF_LUID和**NET_LUID_LH**结构是可用于**NET_LUID**联合的其他名称。

IfType 位域的值在 *Iplifcons.h* 包含文件中定义。当前仅支持 **IfType** 成员的说明中列出的可能值。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]

另请参阅

[ConvertInterfaceAliasToLuid](#)

[ConvertInterfaceGuidToLuid](#)

[ConvertInterfaceIndexToLuid](#)

[ConvertInterfaceLuidToGuid](#)

[ConvertInterfaceLuidToIndex](#)

[ConvertInterfaceLuidToNameA](#)

[ConvertInterfaceLuidToNameW](#)

[ConvertInterfaceNameToLuidA](#)

[ConvertInterfaceNameToLuidW](#)

[MIB_IF_ROW2](#)

[MIB_IPINTERFACE_ROW](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

SOCKADDR_IN6_PAIR 结构 (ws2ipdef.h)

项目2023/08/28

SOCKADDR_IN6_PAIR 结构包含指向表示源地址对和目标地址对的 IP 地址对的指针。

语法

C++

```
typedef struct _sockaddr_in6_pair {
    PSOCKADDR_IN6 SourceAddress;
    PSOCKADDR_IN6 DestinationAddress;
} SOCKADDR_IN6_PAIR, *PSOCKADDR_IN6_PAIR;
```

成员

SourceAddress

指向表示为SOCKADDR_IN6结构的 IP 源地址 的 指针。 地址系列采用主机字节顺序，IPv6 地址、端口、流信息和区域 ID 按网络字节顺序排列。

DestinationAddress

指向表示为SOCKADDR_IN6结构的 IP 源地址 的 指针。 地址系列采用主机字节顺序，IPv6 地址、端口、流信息和区域 ID 按网络字节顺序排列。

注解

SOCKADDR_IN6_PAIR结构在 Windows Vista 及更高版本上定义。

SOCKADDR_IN6_PAIR结构中的任何 IPv4 地址都必须以 IPv4 映射的 IPv6 地址格式表示，这样，仅限 IPv6 的应用程序才能与 IPv4 节点通信。有关 IPv4 映射的 IPv6 地址格式的详细信息，请参阅 [双堆栈套接字](#)。

SOCKADDR_IN6_PAIR 结构由 [CreateSortedAddressPairs](#) 函数使用。

请注意， *Ws2ipdef.h* 头文件会自动包含在 *Ws2tcpip.h* 头文件中，永远不应直接使用。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	ws2ipdef.h (包括 Ws2tcpip.h)

另请参阅

[CreateSortedAddressPairs](#)

[双堆栈套接字](#)

[sockaddr](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

NL_BANDWIDTH_INFORMATION 结构 (nldef.h)

项目2023/08/25

NL_BANDWIDTH_INFORMATION 结构包含有关可用带宽估计和相关差异的只读信息，由 TCP/IP 堆栈确定。

语法

C++

```
typedef struct _NL_BANDWIDTH_INFORMATION {
    ULONG64 Bandwidth;
    ULONG64 Instability;
    BOOLEAN BandwidthPeaked;
} NL_BANDWIDTH_INFORMATION, *PNL_BANDWIDTH_INFORMATION;
```

成员

Bandwidth

估计的最大可用带宽（以位/秒为单位）。

Instability

基于最近带宽样本的变化度量值，以位/秒为单位。

BandwidthPeaked

一个值，该值指示 **带宽** 成员中的带宽估计是否已达到峰值，并达到给定网络条件的最大值。

TCP/IP 堆栈使用启发法设置此变量。在设置此变量之前，无法保证真正的可用最大带宽不高于 **Bandwidth** 成员中的估计带宽。但是，可以放心地假设最大可用带宽不低于 **带宽** 成员中报告的估计值。

注解

NL_BANDWIDTH_INFORMATION 结构在 *Nldef.h* 头文件中定义，该文件由 *Iphlpapi.h* 头文件中自动包含的 *Iptypes.h* 头文件自动包含。永远不应直接使用 *Nldef.h* 和 *Iptypes.h* 头

文件。

要求

最低受支持的客户端	Windows 8 [仅限桌面应用]
最低受支持的服务器	Windows Server 2012 [仅限桌面应用]
标头	nldef.h (包括 Iphlpapi.h)

请参阅

[GetIpNetworkConnectionBandwidthEstimates](#)

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[MIB_IP_NETWORK_CONNECTION_BANDWIDTH_ESTIMATES](#)

[TCP_ESTATS_BANDWIDTH_ROD_v0](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

SOCKADDR_INET union (ws2ipdef.h)

项目2023/08/28

SOCKADDR_INET联合包含 IPv4、IPv6 地址或地址系列。

语法

C++

```
typedef union _SOCKADDR_INET {
    SOCKADDR_IN     Ipv4;
    SOCKADDR_IN6    Ipv6;
    ADDRESS_FAMILY  si_family;
} SOCKADDR_INET, *PSOCKADDR_INET;
```

成员

Ipv4

类型: SOCKADDR_IN

表示为包含地址系列和 [IPv4 地址的SOCKADDR_IN](#) 结构的 IPv4 地址。 地址系列采用主机字节顺序，IPv4 地址采用网络字节顺序。

在为 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织已更改，[SOCKADDR_IN](#) 结构在 *Ws2def.h* 头文件中定义。 请注意，*Ws2def.h* 头文件自动包含在 *Winsock2.h* 中，不应直接使用。

Ipv6

类型: SOCKADDR_IN6

表示为包含地址系列和 [IPv6 地址的SOCKADDR_IN6](#) 结构的 IPv6 地址。 地址系列采用主机字节顺序，IPv6 地址采用网络字节顺序。

在为 Windows Vista 及更高版本发布的 Windows SDK 上，头文件的组织已更改，[SOCKADDR_IN6](#) 结构在 *Ws2def.h* 头文件中定义。 请注意，*Ws2def.h* 头文件自动包含在 *Winsock2.h* 中，不应直接使用。

si_family

类型: ADDRESS_FAMILY

地址系列。

Ws2def.h 头文件中列出了地址系列的可能值。请注意，AF_ 地址系列和PF_ 协议系列常量的值 (相同，例如，AF_INET 和 PF_INET)，因此可以使用任一常量。*Ws2def.h* 头文件自动包含在 *Winsock2.h* 中，不应直接使用。

当前支持的值是 AF_INET、AF_INET6 和 AF_UNSPEC。

Value	含义
AF_UNSPEC 0	未指定地址系列。指定此参数后， <code>SOCKADDR_INET</code> 联合可以表示 IPv4 或 IPv6 地址系列。
AF_INET 2	Internet 协议版本 4 (IPv4) 地址系列。
AF_INET6 23	Internet 协议版本 6 (IPv6) 地址系列。

注解

`SOCKADDR_INET` 联合在 Windows Vista 及更高版本上定义。

`SOCKADDR_INET` 联合是一种方便的结构，用于访问 IPv4 地址、IPv6 地址或 IP 地址系列，而无需强制转换 `sockaddr` 结构。

`SOCKADDR_INET` 联合是 [IP_ADDRESS_PREFIX](#) 结构中 `Prefix` 成员的数据类型

请注意，*Ws2ipdef.h* 头文件自动包含在 *Ws2tcpip.h* 头文件中，不应直接使用。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	<code>ws2ipdef.h</code> (包括 <code>Ws2tcpip.h</code>)

另请参阅

[IP_ADDRESS_PREFIX](#)

[sockaddr](#)

反馈

此页面是否有帮助?

是

否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_BANDWIDTH_ROD_v0结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_BANDWIDTH_ROD_v0 结构包含有关 TCP 连接的带宽估计的扩展 TCP 统计信息的只读动态信息。

语法

C++

```
typedef struct _TCP_ESTATS_BANDWIDTH_ROD_v0 {
    ULONG64 OutboundBandwidth;
    ULONG64 InboundBandwidth;
    ULONG64 OutboundInstability;
    ULONG64 InboundInstability;
    BOOLEAN OutboundBandwidthPeaked;
    BOOLEAN InboundBandwidthPeaked;
} TCP_ESTATS_BANDWIDTH_ROD_v0, *PTCP_ESTATS_BANDWIDTH_ROD_v0;
```

成员

OutboundBandwidth

类型： ULONG64

TCP 连接的网络路径的计算出站带宽估计值（以位数/秒为单位）。

InboundBandwidth

类型： ULONG64

计算的 TCP 连接的网络路径的入站带宽估计值（以位数/秒为单位）。

OutboundInstability

类型： ULONG64

以位/秒为单位的 TCP 连接的网络路径的出站带宽估计不稳定的度量值。

InboundInstability

类型： ULONG64

以位数/秒为单位的 TCP 连接的网络路径的入站带宽估计不稳定的度量值。

`OutboundBandwidthPeaked`

类型: **BOOLEAN**

一个布尔值，指示 TCP 连接的网络路径的计算出站带宽估计是否已达到其峰值。

`InboundBandwidthPeaked`

类型: **BOOLEAN**

一个布尔值，指示 TCP 连接的网络路径的计算入站带宽估计是否已达到其峰值。

注解

`TCP_ESTATS_BANDWIDTH_ROD_v0` 结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

`TCP_ESTATS_BANDWIDTH_ROD_v0` 定义为结构版本 0，用于获取有关 TCP 连接的带宽估计的扩展 TCP 统计信息的只读动态信息。建立连接后，此信息可用。

在 `EstatsType` 参数中传递 `TcpConnectionEstatsBandwidth` 时，通过调用 `GetPerTcp6ConnectionEStats` 或 `GetPerTcpConnectionEStats` 函数来检索 `TCP_ESTATS_BANDWIDTH_ROD_v0` 结构。需要启用扩展 TCP 统计信息才能检索此结构。

此结构的成员未在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。有关此 RFC 的详细信息，请参阅 <http://www.ietf.org/rfc/rfc4898.txt>。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcppestats.h

另请参阅

[GetIpNetworkConnectionBandwidthEstimates](#)

[GetPerTcp6ConnectionEStats](#)

GetPerTcpConnectionEStats

NL_BANDWIDTH_INFORMATION

TCP_ESTATS_TYPE

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_BANDWIDTH_RW_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_BANDWIDTH_RW_v0 结构包含有关 TCP 连接的带宽估计的扩展 TCP 统计信息的读/写配置信息。

语法

C++

```
typedef struct _TCP_ESTATS_BANDWIDTH_RW_v0 {
    TCP_BOOLEAN_OPTIONAL EnableCollectionOutbound;
    TCP_BOOLEAN_OPTIONAL EnableCollectionInbound;
} TCP_ESTATS_BANDWIDTH_RW_v0, *PTCP_ESTATS_BANDWIDTH_RW_v0;
```

成员

EnableCollectionOutbound

一个值，该值指示是否应收集 TCP 连接上的扩展统计信息以估计出站带宽。

如果此成员设置为 `TcpBoolOptEnabled`，则会启用 TCP 连接上的扩展统计信息，用于估计出站带宽。如果此成员设置为 `TcpBoolOptDisabled`，则会禁用 TCP 连接的扩展统计信息，以便估计出站带宽。如果此成员设置为 `TcpBoolOptUnchanged`，则出站带宽估算的 TCP 连接的扩展统计信息保持不变。

未设置时此成员的默认状态处于禁用状态。

EnableCollectionInbound

一个值，该值指示是否应收集 TCP 连接上的扩展统计信息来估算入站带宽。

如果此成员设置为 `TcpBoolOptEnabled`，则会启用 TCP 连接的扩展统计信息，以便估算入站带宽。如果此成员设置为 `TcpBoolOptDisabled`，则会禁用 TCP 连接上的扩展统计信息，以便估算入站带宽。如果此成员设置为 `TcpBoolOptUnchanged`，则入站带宽估算的 TCP 连接扩展统计信息保持不变。

未设置时此成员的默认状态处于禁用状态。

注解

`TCP_ESTATS_BANDWIDTH_RW_v0` 结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

`TCP_ESTATS_BANDWIDTH_RW_v0` 定义为结构版本 0，用于提供有关 TCP 连接的带宽估算的读/写配置信息。

在 `EstatsType` 参数中传递 `TcpConnectionEstatsBandwidth` 时，会使用此结构以及 `SetPerTcp6ConnectionEStats` 和 `SetPerTcpConnectionEStats` 函数启用和禁用 TCP 连接的带宽估算扩展 TCP 统计信息。

传递给 `SetPerTcp6ConnectionEStats` 和 `SetPerTcpConnectionEStats` 函数的 `Offset` 参数当前未使用，必须设置为 0。因此，当 `EstatsType` 参数设置为 `TcpConnectionEstatsBandwidth` 时，`Rw` 参数指向的 `TCP_ESTATS_BANDWIDTH_RW_v0` 结构必须将 `EnableCollectionOutbound` 和 `EnableCollectionInbound` 结构成员设置为对 `SetPerTcp6ConnectionEStats` 和 `SetPerTcpConnectionEStats` 函数的单个调用中的首选值。

在 `EstatsType` 参数中传递 `TcpConnectionEstatsBandwidth` 时，将通过调用 `GetPerTcp6ConnectionEStats` 或 `GetPerTcpConnectionEStats` 函数来检索 `TCP_ESTATS_BANDWIDTH_RW_v0` 结构。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcppestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_BOOLEAN_OPTIONAL](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

tcpestats.h) (TCP_ESTATS_DATA_ROD_v0 结构

项目2023/08/26

TCP_ESTATS_DATA_ROD_v0 结构包含有关 TCP 连接的数据传输的扩展 TCP 统计信息的只读动态信息。

语法

C++

```
typedef struct _TCP_ESTATS_DATA_ROD_v0 {
    ULONG64 DataBytesOut;
    ULONG64 DataSegsOut;
    ULONG64 DataBytesIn;
    ULONG64 DataSegsIn;
    ULONG64 SegsOut;
    ULONG64 SegsIn;
    ULONG    SoftErrors;
    ULONG    SoftErrorReason;
    ULONG    SndUna;
    ULONG    SndNxt;
    ULONG    SndMax;
    ULONG64 ThruBytesAcked;
    ULONG    RcvNxt;
    ULONG64 ThruBytesReceived;
} TCP_ESTATS_DATA_ROD_v0, *PTCP_ESTATS_DATA_ROD_v0;
```

成员

DataBytesOut

类型: **ULONG64**

传输段中包含的数据的八位字节数，包括重新传输的数据。请注意，这不包括 TCP 标头。

DataSegsOut

类型: **ULONG64**

发送的包含正长度数据段的段数。

DataBytesIn

类型: **ULONG64**

接收的数据段中包含的八进制数，包括重新传输的数据。请注意，这不包括 TCP 标头。

DataSegsIn

类型: **ULONG64**

收到的包含正长度数据段的段数。

SegsOut

类型: **ULONG64**

发送的段总数。

SegsIn

类型:

收到的段总数。

SoftErrors

类型: **ULONG**

TCP 输入处理期间未通过各种一致性测试的段数。软错误可能会导致段被丢弃，但有些则不然。其中一些软错误会导致生成 TCP 确认，而其他错误则以无提示方式丢弃。

SoftErrorReason

类型: **ULONG**

一个值，该值标识在 TCP 输入处理期间最近失败的一致性测试。每次递增 **SoftErrors** 成员时都会设置此对象。

SndUna

类型: **ULONG**

最早的未确认序列号的值。请注意，此成员是 TCP 状态变量。

SndNxt

类型: **ULONG**

要发送的下一个序列号。请注意，此成员不是单调（因此不是计数器），因为 TCP 有时会通过将成员拉回到缺失的数据来重新传输丢失的数据。

SndMax

类型: ULONG

最远的前向 (最右边或最大的) 要发送的序列号。请注意，这将等于 SndNxt 成员，除非在恢复期间拉回 SndNxt 成员。

ThruBytesAcked

类型: ULONG64

已收到累积确认的八进制数。请注意，这是 SndNxt 成员的更改总和。

RcvNxt

类型: ULONG

要接收的下一个序列号。请注意，此成员不是单调 (因此不是计数器)，因为 TCP 有时会通过将成员拉回到缺失的数据来重新传输丢失的数据。

ThruBytesReceived

类型: ULONG64

已为其发送累积确认的八进制数。请注意，这是 对 RcvNxt 成员的更改总和。

注解

TCP_ESTATS_DATA_ROD_v0 结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_DATA_ROD_v0 定义为结构版本 0，用于获取有关 TCP 连接的数据传输的扩展 TCP 统计信息的只读动态信息。建立连接后，此信息可用。

在 *EstatsType* 参数中传递 TcpConnectionEstatsData 时，通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索 TCP_ESTATS_DATA_ROD_v0 结构。需要启用扩展 TCP 统计信息才能检索此结构。

此结构的成员在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4898.txt>。

下面是 TCP_ESTATS_DATA_ROD_v0 结构中的成员到 RFC 4898 中为扩展 TCP 统计信息定义的条目的映射：

术语	说明
----	----

DataBytesOut	tcpEStatsPerfDataOctetsOut
DataSegsOut	tcpEStatsPerfDataSegsOut
DataBytesIn	tcpEStatsPerfDataOctetsIn
DataSegsIn	tcpEStatsPerfDataSegsIn
SegsOut	tcpEStatsPerfSegsOut
SegsIn	tcpEStatsPerfSegsIn
SoftErrors	tcpEStatsStackSoftErrors
SoftErrorReason	tcpEStatsStackSoftErrorReason
SndUna	tcpEStatsAppSndUna
SndNxt	tcpEStatsAppSndNxt
SndMax	tcpEStatsAppSndMax
ThruBytesAcked	tcpEStatsAppThruOctetsAcked
RcvNxt	tcpEStatsAppRcvNxt
ThruBytesReceived	tcpEStatsAppThruOctetsReceived

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_DATA_RW_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_DATA_RW_v0 结构包含有关 TCP 连接的数据传输的扩展 TCP 统计信息的读/写配置信息。

语法

C++

```
typedef struct _TCP_ESTATS_DATA_RW_v0 {
    BOOLEAN EnableCollection;
} TCP_ESTATS_DATA_RW_v0, *PTCP_ESTATS_DATA_RW_v0;
```

成员

EnableCollection

一个值，该值指示是否应为数据传输信息收集 TCP 连接的扩展统计信息。

如果此成员设置为 TRUE，则会启用 TCP 连接上的扩展统计信息。如果此成员设置为 FALSE，则会禁用 TCP 连接的扩展统计信息。

未设置时此成员的默认状态处于禁用状态。

注解

TCP_ESTATS_DATA_RW_v0 结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_DATA_RW_v0 定义为结构版本 0，用于 TCP 连接的扩展数据传输的读/写配置信息。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsData](#) 时，会使用此结构以及 [SetPerTcp6ConnectionEStats](#) 和 [SetPerTcpConnectionEStats](#) 函数启用和禁用 TCP 连接的扩展数据传输信息的扩展 TCP 统计信息。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsData](#) 时，通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索

TCP_ESTATS_DATA_RW_v0结构。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpstats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

tcpestats.h)

(TCP_ESTATS_FINE_RTT_ROD_v0 结构

项目2023/08/26

TCP_ESTATS_FINE_RTT_ROD_v0 结构包含有关 TCP 连接的精细往返时间 (RTT) 估计的扩展 TCP 统计信息的只读动态信息。

语法

C++

```
typedef struct _TCP_ESTATS_FINE_RTT_ROD_v0 {
    ULONG RttVar;
    ULONG MaxRtt;
    ULONG MinRtt;
    ULONG SumRtt;
} TCP_ESTATS_FINE_RTT_ROD_v0, *PTCP_ESTATS_FINE_RTT_ROD_v0;
```

成员

RttVar

类型: ULONG

启用 TCP 扩展统计信息功能时，接收窗口自动优化中使用的往返时间变化（以微秒为单位）。

MaxRtt

类型: ULONG

最大采样往返时间，以微秒为单位。

MinRtt

类型: ULONG

最小采样往返时间，以微秒为单位。

SumRtt

类型: ULONG

从所有采样的往返时间计算的平滑值往返时间（以微秒为单位）。平滑是使用 RttVar 成员的加权累加函数。

注解

`TCP_ESTATS_FINE_RTT_ROD_v0` 结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

`TCP_ESTATS_FINE_RTT_ROD_v0` 定义为结构版本 0，用于获取 TCP 连接的精细往返时间估算的扩展 TCP 统计信息的只读动态信息。建立连接后，此信息可用。

在 `EstatsType` 参数中传递 `TcpConnectionEstatsFineRtt` 时，通过调用 `GetPerTcp6ConnectionEStats` 或 `GetPerTcpConnectionEStats` 函数来检索 `TCP_ESTATS_FINE_RTT_ROD_v0` 结构。需要启用扩展 TCP 统计信息才能检索此结构。

计算 TCP 的重新传输计时器的 IETF RFC 2988 中详细介绍了 TCP 重新传输计时器。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2988.txt>。

此结构的成员未在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。但是，`TCP_ESTATS_PATH_ROD_v0` 结构中有一些成员提供类似的时间度量（以毫秒为单位）。有关详细信息，请参阅 `TCP_ESTATS_PATH_ROD_v0` 结构和 <http://www.ietf.org/rfc/rfc4898.txt>。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcppestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_PATH_ROD_v0](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_FINE_RTT_RW_v0 结构 (tcpestats.h)

项目2023/08/26

TCP_ESTATS_FINE_RTT_RW_v0 结构包含针对 TCP 连接的细粒度往返时间 (RTT) 估计统计信息的扩展 TCP 统计信息的读/写配置信息。

语法

C++

```
typedef struct _TCP_ESTATS_FINE_RTT_RW_v0 {
    BOOLEAN EnableCollection;
} TCP_ESTATS_FINE_RTT_RW_v0, *PTCP_ESTATS_FINE_RTT_RW_v0;
```

成员

EnableCollection

一个值，该值指示是否应收集 TCP 连接上的扩展统计信息，以便进行精细的 RTT 估计统计信息。

如果此成员设置为 TRUE，则会启用 TCP 连接上的扩展统计信息。如果此成员设置为 FALSE，则会禁用 TCP 连接的扩展统计信息。

未设置时此成员的默认状态处于禁用状态。

注解

TCP_ESTATS_FINE_RTT_RW_v0 结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_FINE_RTT_RW_v0 定义为结构版本 0，用于提供有关 TCP 连接的细粒度往返时间估计统计信息的读/写配置信息。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsFineRtt](#) 时，会使用此结构以及 [SetPerTcp6ConnectionEStats](#) 和 [SetPerTcpConnectionEStats](#) 函数启用和禁用 TCP 连接的扩展路径度量信息的扩展 TCP 统计信息。

在 `EstatsType` 参数中传递 `TcpConnectionEstatsFineRtt` 时，可通过调用 `GetPerTcp6ConnectionEStats` 或 `GetPerTcpConnectionEStats` 函数来检索 `TCP_ESTATS_FINE_RTT_RW_v0` 结构。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcppestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_OBS_REC_ROD_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_OBS_REC_ROD_v0 结构包含针对 TCP 连接的远程接收器上观察到的扩展 TCP 统计信息的只读动态信息。

语法

C++

```
typedef struct _TCP_ESTATS_OBS_REC_ROD_v0 {
    ULONG CurRwinRcvd;
    ULONG MaxRwinRcvd;
    ULONG MinRwinRcvd;
    UCHAR WinScaleRcvd;
} TCP_ESTATS_OBS_REC_ROD_v0, *PTCP_ESTATS_OBS_REC_ROD_v0;
```

成员

CurRwinRcvd

类型: **ULONG**

从远程接收器接收的最新窗口播发 (以字节为单位)。

MaxRwinRcvd

类型: **ULONG**

从远程接收方接收的最大窗口播发 (以字节为单位)。

MinRwinRcvd

类型: **ULONG**

从远程接收方接收的最小窗口播发 (以字节为单位)。

WinScaleRcvd

类型: **ULONG**

接收的窗口缩放选项的值 (如果从远程接收方收到);否则, 值为 -1。

请注意，如果 `TCP_ESTATS_REC_ROD_v0` 结构的 `WinScaleSent` 成员和 `WinScaleRcvd` 成员都不是 -1，则 `Snd.Wind.Scale` 将与此值相同，并用于将接收方窗口通知从远程主机缩放到本地主机。

注解

`TCP_ESTATS_OBS_REC_ROD_v0` 结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

`TCP_ESTATS_OBS_REC_ROD_v0` 定义为结构版本 0，用于获取 TCP 连接的本地接收方上的扩展 TCP 统计信息的只读动态信息。建立连接后，此信息可用。

在 `EstatsType` 参数中传递 `TcpConnectionEstatsObsRec` 时，通过调用 `GetPerTcp6ConnectionEStats` 或 `GetPerTcpConnectionEStats` 函数来检索 `TCP_ESTATS_OBS_REC_ROD_v0` 结构。需要启用扩展 TCP 统计信息才能检索此结构。

此结构的成员在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4898.txt>。

下面是 `TCP_ESTATS_OBS_REC_ROD_v0` 结构中的成员到 RFC 4898 中为扩展 TCP 统计信息定义的条目的映射：

术语	说明
<code>CurRwinRcvd</code>	<code>tcpEStatsPerfCurRwinRcvd</code>
<code>MaxRwinRcvd</code>	<code>tcpEStatsPerfMaxRwinRcvd</code>
<code>MinRwinRcvd</code>	没有映射到此成员。
<code>WinScaleRcvd</code>	<code>tcpEStatsStackWinScaleRcvd</code>

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	<code>tcpestats.h</code>

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_OBS_REC_RW_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_OBS_REC_RW_v0 结构包含针对 TCP 连接的远程接收器上观察到的扩展 TCP 统计信息的读/写配置信息。

语法

C++

```
typedef struct _TCP_ESTATS_OBS_REC_RW_v0 {
    BOOLEAN EnableCollection;
} TCP_ESTATS_OBS_REC_RW_v0, *PTCP_ESTATS_OBS_REC_RW_v0;
```

成员

EnableCollection

一个值，该值指示是否应收集 TCP 连接上的扩展统计信息以获取远程接收器信息。

如果此成员设置为 TRUE，则会启用 TCP 连接上的扩展统计信息。如果此成员设置为 FALSE，则会禁用 TCP 连接的扩展统计信息。

未设置时此成员的默认状态处于禁用状态。

注解

TCP_ESTATS_OBS_REC_RW_v0 结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_OBS_REC_RW_v0 定义为在 TCP 连接的远程接收器上观察到的读/写配置信息的结构版本 0。

在 *EstatsType* 参数中传递 `TcpConnectionEstatsObsRec` 时，会使用此结构以及 `SetPerTcp6ConnectionEStats` 和 `SetPerTcpConnectionEStats` 函数启用和禁用 TCP 连接的远程接收方信息的扩展 TCP 统计信息。

在 *EstatsType* 参数中传递 `TcpConnectionEstatsObsRec` 时，将通过调用 `GetPerTcp6ConnectionEStats` 或 `GetPerTcpConnectionEStats` 函数来检索

TCP_ESTATS_REC_RW_v0结构。

要求

最低受支持的客户端	
	Windows Vista [仅限桌面应用]
最低受支持的服务器	
	Windows Server 2008 [仅限桌面应用]
标头	
	tcpstats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

tcpestats.h) (TCP_ESTATS_PATH_ROD_v0 结构

项目2023/08/26

TCP_ESTATS_PATH_ROD_v0结构包含有关 TCP 连接的网络路径度量的扩展 TCP 统计信息的只读动态信息。

语法

C++

```
typedef struct _TCP_ESTATS_PATH_ROD_v0 {
    ULONG FastRetran;
    ULONG Timeouts;
    ULONG SubsequentTimeouts;
    ULONG CurTimeoutCount;
    ULONG AbruptTimeouts;
    ULONG PktsRetrans;
    ULONG BytesRetrans;
    ULONG DupAcksIn;
    ULONG SacksRcvd;
    ULONG SackBlocksRcvd;
    ULONG CongSignals;
    ULONG PreCongSumCwnd;
    ULONG PreCongSumRtt;
    ULONG PostCongSumRtt;
    ULONG PostCongCountRtt;
    ULONG EcnSignals;
    ULONG EceRcvd;
    ULONG SendStall;
    ULONG QuenchRcvd;
    ULONG RetranThresh;
    ULONG SndDupAckEpisodes;
    ULONG SumBytesReordered;
    ULONG NonRecovDa;
    ULONG NonRecovDaEpisodes;
    ULONG AckAfterFr;
    ULONG DsackDups;
    ULONG SampleRtt;
    ULONG SmoothedRtt;
    ULONG RttVar;
    ULONG MaxRtt;
    ULONG MinRtt;
    ULONG SumRtt;
    ULONG CountRtt;
    ULONG CurRto;
    ULONG MaxRto;
    ULONG MinRto;
```

```
    ULONG CurMss;
    ULONG MaxMss;
    ULONG MinMss;
    ULONG SpuriousRtoDetections;
} TCP_ESTATS_PATH_ROD_v0, *PTCP_ESTATS_PATH_ROD_v0;
```

成员

FastRetran

类型: **ULONG**

快速重新传输算法的调用次数。

Timeouts

类型: **ULONG**

当重新传输计时器退避乘数等于 1 时，重新传输超时已过期的次数。

SubsequentTimeouts

类型: **ULONG**

重新传输计时器翻倍后，重新传输超时已过期的次数。

有关详细信息，请参阅以下备注中讨论的 RFC 2988 的第 5.5 节。

CurTimeoutCount

类型: **ULONG**

重新传输超时已过期且未收到新数据的确认的当前次数。

当对 RFC 2988 第 5.5 节的每次调用确认和递增新数据时，**CurTimeoutCount** 成员将重置为零。

AbruptTimeouts

类型: **ULONG**

在没有任何重复确认或其他拥塞迹象的情况下发生的超时次数。突然超时表示路径丢失了整个数据窗口或确认。

(显式拥塞通知之前出现重复确认或其他拥塞信号的超时，例如，) 不计为突然，并且可能已被更复杂的快速重新传输算法所避免。

PktsRetrans

类型: **ULONG**

传输的段数，其中包含至少一些重新传输的数据。

BytesRetrans

类型: **ULONG**

重新传输的字节数。

DupAcksIn

类型: **ULONG**

收到的重复 ACK 数。

SacksRcvd

类型: **ULONG**

收到的选择性确认 (SACK) 选项的数目。

SackBlocksRcvd

类型: **ULONG**

SACK 选项) (收到的 SACK 块数。

CongSignals

类型: **ULONG**

由于各种形式的拥塞信号（包括快速重新传输、显式拥塞通知 (ECN) 和超时）而导致的乘法向下拥塞窗口调整的数目。此成员总结了调用加法增加乘法减少 (MD) 部分的加法增加乘积减少 (AIMD) 拥塞控制的所有事件，因此是拥塞窗口如何受到拥塞影响的最佳指标。

请注意，重新传输超时通过设置慢启动阈值大小以乘法递减窗口，并包含在 CongSignals 成员中存储的值中。为了尽量减少由于无序段而导致的虚假拥塞指示，CongSignals 成员与快速重新传输算法结合使用。

PreCongSumCwnd

类型: **ULONG**

每次收到拥塞信号时捕获的拥塞窗口值的总和（以字节为单位）。

每次递增 CongSignals 成员时都会更新此成员，这样 PreCongSumCwnd 成员中的更改除以 CongSignals 成员中的更改就是在拥塞信号之前某个间隔 (的平均窗口)。

PreCongSumRtt

类型: ULONG

在接收拥塞信号之前，网络往返时间的最后一个样本 (RTT) 的总和 (以毫秒为单位)。RTT 的最后一个示例存储在 SampleRtt 成员中。

每次递增 CongSignals 成员时，PreCongSumRtt 成员都会更新，这样 PreCongSumRtt 中的更改除以 CongSignals 成员中的更改就是在拥塞信号之前某个时间间隔) 的平均 RTT ()。

PostCongSumRtt

类型: ULONG

每个拥塞信号后存储在 SampleRtt 成员) 的网络 RTT (的第一个样本的总和 (以毫秒为单位)。

PostCongSumRtt 成员中的更改除以 PostCongCountRtt 成员中的更改，是某个时间间隔内的平均 RTT () 紧接拥塞信号之后。

PostCongCountRtt

类型: ULONG

PostCongSumRtt 成员中包含的 RTT 样本数 (以字节为单位)。

PostCongSumRtt 成员中的更改除以 PostCongCountRtt 成员中的更改，是某个时间间隔内的平均 RTT () 紧接拥塞信号之后。

EcnSignals

类型: ULONG

通过 ECN 传递到 TCP 发送方的拥塞信号数。

这通常是包含 Echo 拥塞的段数

经验丰富的 (ECE) 位，但也包括无法通过 ECN nonce 检查或其他显式拥塞信号的段。

EceRcvd

类型: ULONG

接收的段数，其中 IP 标头带有 ce) 标记的拥塞体验 (。

`SendStall`

类型: **ULONG**

被视为拥塞信号的接口停止数或其他发送方本地资源限制。

`QuenchRcvd`

类型: **ULONG**

保留供将来使用。此成员始终设置为零。

`RetranThresh`

类型: **ULONG**

触发快速重新传输所需的重复确认数。

请注意，尽管这在传统的 Reno TCP 实现中是恒定的，但它在许多较新的 TCP 实现中是自适应的。

`SndDupAckEpisodes`

类型: **ULONG**

上一个确认不重复时发送的重复确认数。这是连续一系列重复确认已发送的次数。

这表示从远程 TCP 终结点到附近 TCP 终结点的路径上丢失或重新排序的数据段数。

`SumBytesReordered`

类型: **ULONG**

SND 的金额之和。UNA 推进确认，结束了一个窃听情节，没有转播。

请注意，`SumBytesReordered` 成员中的更改除以 `NonRecovDaEpisodes` 成员中的更改是某个时间间隔内平均重新排序距离的估计值。

`NonRecovDa`

类型: **ULONG**

由于 ACK 先于到达 `RetranThresh` 的重复确认数，(或 SACKS) 未触发快速重新传输。

请注意，`NonRecovDa` 成员的更改除以 `NonRecovDaEpisodes` 成员中的更改是段内某个间隔的平均重新排序距离的估计值。

`NonRecovDaEpisodes`

类型: **ULONG**

由于 ACK 先于到达 RetranThresh 的重复确认数，因此未触发快速重新传输的重复确认剧集数。

AckAfterFr

类型: **ULONG**

保留供将来使用。此成员始终设置为零。

DsackDups

类型: **ULONG**

D-SACK 块向本地主机报告的重复段数。

SampleRtt

类型: **ULONG**

用于计算重新传输计时器 (RTO) 的最新原始网络往返时间度量 (以毫秒为单位)。

SmoothedRtt

类型: **ULONG**

用于计算 RTO 的平滑往返时间 (以毫秒为单位)。

RttVar

类型: **ULONG**

用于计算 RTO 的往返时间变化 (以毫秒为单位)。

MaxRtt

类型: **ULONG**

最大采样往返时间 (以毫秒为单位)。

MinRtt

类型: **ULONG**

最小采样往返时间 (以毫秒为单位)。

SumRtt

类型: **ULONG**

所有采样的往返行程时间之和 (以毫秒为单位)。

请注意, **SumRtt** 成员中的更改除以 **CountRtt** 成员中的更改是平均 RTT, 在输入间隔内统一求平均值。

CountRtt

类型: **ULONG**

SumRtt 成员中包含的往返时间示例数。

CurRto

类型: **ULONG**

重新传输计时器的当前值 (以毫秒为单位)。

MaxRto

类型: **ULONG**

重新传输计时器的最大值 (以毫秒为单位)。

MinRto

类型: **ULONG**

重新传输计时器的最小值 (以毫秒为单位)。

CurMss

类型: **ULONG**

当前最大段大小 (MSS) (以字节为单位)。

MaxMss

类型: **ULONG**

最大 MSS, 以字节为单位。

MinMss

类型: **ULONG**

最小 MSS, 以字节为单位。

类型: ULONG

由于重新传输超时而已重新传输的确认报告段的数目。

注解

TCP_ESTATS_PATH_ROD_v0 结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_PATH_ROD_v0 定义为结构版本 0，用于获取有关 TCP 连接的网络路径度量的只读动态信息。 建立连接后，此信息可用。

在 EstatsType 参数中传递 [TcpConnectionEstatsPath](#) 时，将通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索 TCP_ESTATS_PATH_ROD_v0 结构。 需要启用扩展 TCP 统计信息才能检索此结构。

路径 MTU 发现中的 IETF RFC 1191 详细讨论了路径 MTU 发现和最大段大小。 有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc1191.txt>。

TCP 拥塞控制上的 IETF RFC 2581 中详细介绍了 TCP 拥塞控制和拥塞控制算法。 有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2581.txt>。

在针对 TCP 的选择性确认扩展的 IETF RFC 2883 中详细讨论了 SACK 和 SACK 选项的扩展 (SACK) 选项。 有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2883.txt>。

计算 TCP 重新传输计时器 (的 IETF RFC 2988 中详细讨论了 RTO) 和平滑的往返时间 (RTT)。 有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2988.txt>。

IETF RFC 2581 中详细介绍了 IP 中的显式拥塞通知 (ECN)。 有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc3168.txt>。

此结构的成员在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。 有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4898.txt>。

下面是 将 TCP_ESTATS_PATH_ROD_v0 结构中的成员映射到 RFC 4898 中为扩展 TCP 统计信息定义的条目：

术语	说明
FastRetran	tcpEStatsStackFastRetran
超时	tcpEStatsPerfTimeouts
SubsequentTimeouts	tcpEStatsStackSubsequentTimeouts

CurTimeoutCount	tcpEStatsStackCurTimeoutCount
AbruptTimeouts	tcpEStatsStackAbruptTimeouts
PktsRetrans	tcpEStatsPerfSegsRetrans
BytesRetrans	tcpEStatsPerfOctetsRetrans
DupAcksIn	tcpEStatsStackDupAcksIn
SacksRcvd	tcpEStatsStackSACKsRcvd
SackBlocksRcvd	tcpEStatsStackSACKBlocksRcvd
CongSignals	tcpEStatsPerfCongSignals
PreCongSumCwnd	tcpEStatsPathPreCongSumCwnd
PreCongSumRtt	tcpEStatsPathPreCongSumRTT
PostCongSumRtt	tcpEStatsPathPostCongSumRTT
PostCongCountRtt	tcpEStatsPathPostCongCountRTT
EcnSignals	tcpEStatsPathECNsignals
EceRcvd	tcpEStatsPathCERcvd
SendStall	tcpEStatsStackSendStall
QuenchRcvd	不映射到此成员。
RetranThresh	tcpEStatsPathRetranThresh
SndDupAckEpisodes	tcpEStatsPathDupAckEpisodes
SumBytesReordered	tcpEStatsPathSumOctetsReordered
NonRecovDa	tcpEStatsPathNonRecovDA
NonRecovDaEpisodes	tcpEStatsPathNonRecovDAEpisodes
AckAfterFr	不映射到此成员。
DsackDups	tcpEStatsStackDSACKDups
SampleRtt	tcpEStatsPathSampleRTT
SmoothedRtt	tcpEStatsPerfSmoothedRTT
RttVar	tcpEStatsPathRTTVar
MaxRtt	tcpEStatsPathMaxRTT

MinRtt	tcpEStatsPathMinRTT
SumRtt	tcpEStatsPathSumRTT
CountRtt	tcpEStatsPathCountRTT
CurRto	tcpEStatsPerfCurRTO
MaxRto	tcpEStatsPathMaxRTO
MinRto	tcpEStatsPathMinRTO
CurMss	tcpEStatsPerfCurMSS
MaxMss	tcpEStatsStackMaxMSS
MinMss	tcpEStatsStackMinMSS
SpuriousRtoDetections	tcpEStatsStackSpuriousRtoDetected

[TCP_ESTATS_FINE_RTT_ROD_v0](#) 结构的成员提供与 [TCP_ESTATS_PATH_ROD_v0](#) 结构的 RttVar、 MaxRtt、 MinRtt 和 SumRtt 成员类似的数据。但是，对于 [TCP_ESTATS_FINE_RTT_ROD_v0](#) 结构的类似成员，报告的时间以微秒为单位。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_FINE_RTT_ROD_v0](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_PATH_RW_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_PATH_RW_v0 结构包含有关 TCP 连接的路径度量的扩展 TCP 统计信息的读/写配置信息。

语法

C++

```
typedef struct _TCP_ESTATS_PATH_RW_v0 {
    BOOLEAN EnableCollection;
} TCP_ESTATS_PATH_RW_v0, *PTCP_ESTATS_PATH_RW_v0;
```

成员

EnableCollection

一个值，该值指示是否应收集 TCP 连接上的扩展统计信息以获取路径度量信息。

如果此成员设置为 TRUE，则会启用 TCP 连接上的扩展统计信息。如果此成员设置为 FALSE，则会禁用 TCP 连接的扩展统计信息。

未设置时此成员的默认状态处于禁用状态。

注解

TCP_ESTATS_PATH_RW_v0 结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_PATH_RW_v0 定义为结构版本 0，用于有关 TCP 连接的扩展路径度量的读/写配置信息。此信息用于推断从本地发送方到远程接收方的路径上的段重新排序。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsPath](#) 时，将使用此结构以及 [SetPerTcp6ConnectionEStats](#) 和 [SetPerTcpConnectionEStats](#) 函数启用和禁用 TCP 连接的扩展路径度量信息的扩展 TCP 统计信息。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsPath](#) 时，通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索

TCP_ESTATS_PATH_RW_v0结构。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpstats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

tcpestats.h) (TCP_ESTATS_REC_ROD_v0 结构

项目2023/08/26

TCP_ESTATS_REC_ROD_v0 结构包含 TCP 连接的本地接收方上的扩展 TCP 统计信息的只读动态信息。

语法

C++

```
typedef struct _TCP_ESTATS_REC_ROD_v0 {
    ULONG CurRwinSent;
    ULONG MaxRwinSent;
    ULONG MinRwinSent;
    ULONG LimRwin;
    ULONG DupAckEpisodes;
    ULONG DupAcksOut;
    ULONG CeRcvd;
    ULONG EcnSent;
    ULONG EcnNoncesRcvd;
    ULONG CurReasmQueue;
    ULONG MaxReasmQueue;
    SIZE_T CurAppRQueue;
    SIZE_T MaxAppRQueue;
    UCHAR WinScaleSent;
} TCP_ESTATS_REC_ROD_v0, *PTCP_ESTATS_REC_ROD_v0;
```

成员

CurRwinSent

类型: **ULONG**

已发送的最新窗口播发 (以字节为单位) 。

MaxRwinSent

类型: **ULONG**

已发送的最大窗口播发 (以字节为单位) 。

MinRwinSent

类型: **ULONG**

已发送的最小窗口广播 (以字节为单位)。

LimRwin

类型: **ULONG**

可以发送的最大窗口广播 (以字节为单位)。

DupAckEpisodes

类型: **ULONG**

上一个确认不重复时发送的重复确认数。这是连续一系列重复确认已发送的次数。

这表示从远程 TCP 终结点到附近 TCP 终结点的路径上丢失或重新排序的数据段数。

DupAcksOut

类型: **ULONG**

发送的重复 ACK 数。

DupAcksOut 成员中的更改与 **DupAckEpisodes** 成员中更改的比率表示在某个时间间隔内重新排序或恢复距离。

CeRcvd

类型: **ULONG**

接收的段数，其中 IP 标头带有 ce) 标记的拥塞体验 (。

EcnSent

类型: **ULONG**

保留供将来使用。此成员始终设置为零。

EcnNoncesRcvd

类型: **ULONG**

保留供将来使用。此成员始终设置为零。

CurReasmQueue

类型: **ULONG**

重新组合队列跨越的序列空间的当前字节数。

这通常是 rcv.nxt 与重组队列最右侧边缘的序列号之间的差异。

MaxReasmQueue

类型: **ULONG**

重新组合队列跨越的序列空间的最大字节数。

这是 **CurReasmQueue** 成员的最大值。

CurAppRQueue

类型: **SIZE_T**

已由 TCP 确认但尚未传递到应用程序的应用程序数据的当前字节数。

MaxAppRQueue

类型: **SIZE_T**

已由 TCP 确认但尚未传递到应用程序的应用程序数据的最大字节数。

WinScaleSent

类型: **UCHAR**

传输的窗口缩放选项的值 (如果已发送) ;否则, 值为 -1。

请注意, 如果 [TCP_ESTATS_OBS_REC_ROD_v0](#) 结构的 **WinScaleSent** 成员和 **WinScaleRcvd** 成员都不是 -1, 则 Rcv.Wind.Scale 将与此值相同, 并用于将接收方窗口通知从本地主机缩放到远程主机。

注解

TCP_ESTATS_REC_ROD_v0 结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_REC_ROD_v0 定义为结构版本 0, 用于获取 TCP 连接的本地接收方上的扩展 TCP 统计信息的只读动态信息。建立连接后, 此信息可用。

在 [EstatsType](#) 参数中传递 [TcpConnectionEstatsRec](#) 时, 通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索 **TCP_ESTATS_REC_ROD_v0** 结构。需要启用扩展 TCP 统计信息才能检索此结构。

TCP 拥塞控制和拥塞控制算法在 TCP 拥塞控制上的 IETF RFC 2581 中进行了详细介绍。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2581.txt>。

IETF RFC 2581 中详细介绍了 IP 中的显式拥塞通知 (ECN)。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc3168.txt>。

此结构的成员在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4898.txt>。

下面是将 `TCP_ESTATS_REC_ROD_v0` 结构中的成员映射到 RFC 4898 中为扩展 TCP 统计信息定义的条目：

术语	说明
<code>CurrwinSent</code>	<code>tcpEStatsPerfCurRwinSent</code>
<code>MaxRwinSent</code>	<code>tcpEStatsPerfMaxRwinSent</code>
<code>MinrwinSent</code>	没有映射到此成员。
<code>LimRwin</code>	<code>tcpEStatsTuneLimRwin</code>
<code>DupAckEpisodes</code>	<code>tcpEStatsPathDupAckEpisodes</code>
<code>DupAcksOut</code>	<code>tcpEStatsPathDupAcksOut</code>
<code>CeRcvd</code>	<code>tcpEStatsPathCERcvd</code>
<code>EcnSent</code>	没有映射到此成员。
<code>EcnNoncesRcvd</code>	没有映射到此成员。
<code>CurReasmQueue</code>	<code>tcpEStatsStackCurReasmQueue</code>
<code>MaxReasmQueue</code>	<code>tcpEStatsStackMaxReasmQueue</code>
<code>CurAppRQueue</code>	<code>tcpEStatsAppCurAppRQueue</code>
<code>MaxAppRQueue</code>	<code>tcpEStatsAppMaxAppRQueue</code>
<code>WinScaleSent</code>	<code>tcpEStatsStackWinScaleSent</code>

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]

标头

tcpestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

TCP_ESTATS_REC_RW_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_REC_RW_v0 结构包含 TCP 连接的本地接收方上的扩展 TCP 统计信息的读/写配置信息。

语法

C++

```
typedef struct _TCP_ESTATS_REC_RW_v0 {
    BOOLEAN EnableCollection;
} TCP_ESTATS_REC_RW_v0, *PTCP_ESTATS_REC_RW_v0;
```

成员

EnableCollection

一个值，该值指示是否应收集 TCP 连接上的扩展统计信息以获取本地接收方信息。

如果此成员设置为 TRUE，则会启用 TCP 连接上的扩展统计信息。如果此成员设置为 FALSE，则会禁用 TCP 连接的扩展统计信息。

未设置时此成员的默认状态处于禁用状态。

注解

TCP_ESTATS_REC_RW_v0 结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_REC_RW_v0 定义为结构版本 0，用于 TCP 连接的本地接收方上的读/写配置信息。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsRec](#) 时，将使用此结构以及 [SetPerTcp6ConnectionEStats](#) 和 [SetPerTcpConnectionEStats](#) 函数启用和禁用 TCP 连接的本地接收方信息的扩展 TCP 统计信息。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsRec](#) 时，通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索

TCP_ESTATS_REC_RW_v0结构。

要求

最低受支持的客户端	
	Windows Vista [仅限桌面应用]
最低受支持的服务器	
	Windows Server 2008 [仅限桌面应用]
标头	
	tcpstats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

tcppestats.h) (TCP_ESTATS_SEND_BUFF_ROD_v0 结构

项目2023/08/26

TCP_ESTATS_SEND_BUFF_ROD_v0 结构包含有关 TCP 连接的输出队列的扩展 TCP 统计信息的只读动态信息。

语法

C++

```
typedef struct _TCP_ESTATS_SEND_BUFF_ROD_v0 {
    SIZE_T CurRetxQueue;
    SIZE_T MaxRetxQueue;
    SIZE_T CurAppWQueue;
    SIZE_T MaxAppWQueue;
} TCP_ESTATS_SEND_BUFF_ROD_v0, *PTCP_ESTATS_SEND_BUFF_ROD_v0;
```

成员

CurRetxQueue

类型: SIZE_T

当前占用重新传输队列的数据字节数。

MaxRetxQueue

类型: SIZE_T

占用重新传输队列的最大数据字节数。

CurAppWQueue

类型: SIZE_T

TCP 缓冲的应用程序数据的当前字节数，等待 SND 左侧 (第一次传输。NXT 或 SndMax)。
。

此数据通常 (和 SND 传输。一旦有可用的拥塞窗口或接收器窗口，NXT 就会向左)。这是随时可用于传输的数据量，无需计划应用程序。如果排队写入数据不足，TCP 性能可能会受到影响。

MaxAppWQueue

类型: SIZE_T

TCP 缓冲的应用程序数据的最大字节数, 等待第一次传输。

这是 CurAppWQueue 成员的最大值。 MaxAppWQueue 和 CurAppWQueue 成员可用于确定排队数据不足是否为稳定状态, (建议队列空间不足) 或暂时性 (指示应用程序性能不足或 CPU 负载过大或计划程序延迟) 。

注解

TCP_ESTATS_SEND_BUFF_ROD_v0 结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_SEND_BUFF_ROD_v0 定义为结构版本 0, 用于获取有关 TCP 连接的输出队列的扩展 TCP 统计信息的只读动态信息。 建立连接后, 此信息可用。

在 `EstatsType` 参数中传递 `TcpConnectionEstatsSendBuff` 时, 通过调用 `GetPerTcp6ConnectionEStats` 或 `GetPerTcpConnectionEStats` 函数来检索 TCP_ESTATS_SEND_BUFF_ROD_v0 结构。 需要启用扩展 TCP 统计信息才能检索此结构。

此结构的成员在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。 有关详细信息, 请参阅 <http://www.ietf.org/rfc/rfc4898.txt> 。

下面是 TCP_ESTATS_SEND_BUFF_ROD_v0 结构中的成员到 RFC 4898 中为扩展 TCP 统计信息定义的条目的映射:

术语	说明
CurRetxQueue	tcpEStatsStackCurRetxQueue
MaxRetxQueue	tcpEStatsStackMaxRetxQueue
CurAppWQueue	tcpEStatsAppCurAppWQueue
MaxAppWQueue	tcpEStatsAppMaxAppWQueue

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]

标头

tcpestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获取帮助](#)

TCP_ESTATS_SEND_BUFF_RW_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_SEND_BUFF_RW_v0 结构包含有关 TCP 连接的输出队列的扩展 TCP 统计信息的读/写配置信息。

语法

C++

```
typedef struct _TCP_ESTATS_SEND_BUFF_RW_v0 {
    BOOLEAN EnableCollection;
} TCP_ESTATS_SEND_BUFF_RW_v0, *PTCP_ESTATS_SEND_BUFF_RW_v0;
```

成员

EnableCollection

一个值，该值指示是否应在输出队列中收集 TCP 连接的扩展统计信息。

如果此成员设置为 TRUE，则会启用 TCP 连接上的扩展统计信息。如果此成员设置为 FALSE，则会禁用 TCP 连接的扩展统计信息。

未设置时此成员的默认状态处于禁用状态。

注解

TCP_ESTATS_SEND_BUFF_RW_v0 结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_SEND_BUFF_RW_v0 定义为有关 TCP 连接的输出队列的读/写配置信息的结构的版本 0。

在 *EstatsType* 参数中传递 *TcpConnectionEstatsSendBuff* 时，会使用此结构以及 *SetPerTcp6ConnectionEStats* 和 *SetPerTcpConnectionEStats* 函数启用和禁用 TCP 连接的输出队列扩展 TCP 统计信息。

在 *EstatsType* 参数中传递 *TcpConnectionEstatsSendBuff* 时，通过调用 *GetPerTcp6ConnectionEStats* 或 *GetPerTcpConnectionEStats* 函数来检索

TCP_ESTATS_SEND_BUFF_RW_v0结构。

要求

最低受支持的客户端	
	Windows Vista [仅限桌面应用]
最低受支持的服务器	
	Windows Server 2008 [仅限桌面应用]
标头	
	tcpstats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

tcppestats.h) (TCP_ESTATS SND_CONG_ROD_v0 结构

项目2023/08/26

TCP_ESTATS SND_CONG_ROD_v0 结构包含有关 TCP 连接的发送方拥塞相关数据的扩展 TCP 统计信息的只读动态信息。

语法

C++

```
typedef struct _TCP_ESTATS_SND_CONG_ROD_v0 {
    ULONG SndLimTransRwin;
    ULONG SndLimTimeRwin;
    SIZE_T SndLimBytesRwin;
    ULONG SndLimTransCwnd;
    ULONG SndLimTimeCwnd;
    SIZE_T SndLimBytesCwnd;
    ULONG SndLimTransSnd;
    ULONG SndLimTimeSnd;
    SIZE_T SndLimBytesSnd;
    ULONG SlowStart;
    ULONG CongAvoid;
    ULONG OtherReductions;
    ULONG CurCwnd;
    ULONG MaxSsCwnd;
    ULONG MaxCaCwnd;
    ULONG CurSsthresh;
    ULONG MaxSsthresh;
    ULONG MinSsthresh;
} TCP_ESTATS_SND_CONG_ROD_v0, *PTCP_ESTATS_SND_CONG_ROD_v0;
```

成员

SndLimTransRwin

类型: ULONG

从“拥塞限制”或“发送方有限”状态转换为“接收方有限”状态的次数。只要 TCP 传输停止，就会进入此状态，因为发送方已填满通知的接收方窗口。

SndLimTimeRwin

类型: ULONG

处于“接收方受限”状态的累积时间（以毫秒为单位），其中 TCP 传输因发送方已填满所宣布的接收方窗口而停止。

`SndLimBytesRwin`

类型: `SIZE_T`

处于“接收方受限”状态的发送的总字节数。

`SndLimTransCwnd`

类型: `ULONG`

从“接收方有限”或“发送方有限”状态转换为“拥塞有限”状态的次数。每当 TCP 传输停止时，就会进入此状态，因为发送方已达到 TCP 拥塞控制（拥塞时段定义的某个限制，例如）或其他算法（重新传输超时）旨在控制网络流量。

`SndLimTimeCwnd`

类型: `ULONG`

处于“拥塞受限”状态的累积时间（以毫秒为单位）。当发生重新传输超时时，该超时将计入此成员中，而不是某个其他状态的累积时间。

`SndLimBytesCwnd`

类型: `SIZE_T`

处于“拥塞受限”状态的发送的总字节数。

`SndLimTransSnd`

类型: `ULONG`

从“接收方有限”或“拥塞有限”状态转换为“发件人有限”状态的次数。每当由于某些发送方限制（例如应用程序数据或其他资源耗尽以及 Karn 算法）而停止 TCP 传输时，就会进入此状态。当 TCP 出于任何原因停止发送数据（不能归类为“接收方受限”或“拥塞限制”）时，它将被视为“发件人有限”。

`SndLimTimeSnd`

类型: `ULONG`

处于“发件人受限”状态的累积时间（以毫秒为单位）。

`SndLimBytesSnd`

类型: **SIZE_T**

处于“发件人限制”状态的发送的总字节数。

SlowStart

类型: **ULONG**

“慢启动”算法增加了拥塞窗口的次数。

CongAvoid

类型: **ULONG**

通过“避免拥塞”算法增加拥塞时段的次数。

OtherReductions

类型: **ULONG**

除“慢启动”和“避免拥塞”算法以外的拥塞控制算法外，其他任何原因导致拥塞窗口减少的数量。

CurCwnd

类型: **ULONG**

当前拥塞窗口的大小 (以字节为单位)。

MaxSsCwnd

类型: **ULONG**

“慢启动”期间使用的拥塞窗口大小的最大大小 (以字节为单位)。

MaxCaCwnd

类型: **ULONG**

“避免拥塞”期间使用的拥塞窗口的最大大小 (以字节为单位)。

CurSsthresh

类型: **ULONG**

慢启动阈值的当前大小 (以字节为单位)。

MaxSsthresh

类型: **ULONG**

慢启动阈值的最大大小 (以字节为单位) , 不包括初始值。

MinSsthresh

类型: **ULONG**

慢启动阈值的最小大小 (以字节为单位) 。

注解

TCP_ESTATS SND CONG ROD v0结构用作 Windows Vista 及更高版本上可用的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS SND CONG ROD v0定义为结构版本 0, 用于获取 TCP 连接的发送方拥塞相关数据的只读动态信息。建立连接后, 此信息可用。

在*EstatsType* 参数中传递 [TcpConnectionEstatsSndCong](#) 时, 通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索 TCP_ESTATS SND CONG ROD v0结构。需要启用扩展 TCP 统计信息才能检索此结构。

TCP 拥塞控制和拥塞控制算法在 TCP 拥塞控制上的 IETF RFC 中进行了详细讨论。有关详细信息, 请参阅 <http://www.ietf.org/rfc/rfc2581.txt> 。

此结构的成员在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。有关详细信息, 请参阅 <http://www.ietf.org/rfc/rfc4898.txt> 。

下面是 TCP_ESTATS SND CONG ROD v0 结构中的成员到 RFC 4898 中为扩展 TCP 统计信息定义的条目的映射:

术语	说明
SndLimTransRwin	tcpEStatsPerfSndLimTransRwin
SndLimTimeRwin	tcpEStatsPerfSndLimTimeRwin
SndLimBytesRwin	没有映射到此成员。
SndLimTransCwnd	tcpEStatsPerfSndLimTransCwnd
SndLimTimeCwnd	tcpEStatsPerfSndLimTimeCwnd
SndLimBytesCwnd	没有映射到此成员。
SndLimTransSnd	tcpEStatsPerfSndLimTransSnd

SndLimTimeSnd	tcpEStatsPerfSndLimTimeSnd
SndLimBytesSnd	没有映射到此成员。
SlowStart	tcpEStatsStackSlowStart
CongAvoid	tcpEStatsStackCongAvoid
OtherReductions	tcpEStatsStackOtherReductions
CurCwnd	tcpEStatsPerfCurCwnd
MaxSsCwnd	tcpEStatsStackMaxSsCwnd
MaxCaCwnd	tcpEStatsStackMaxCaCwnd
CurSsthresh	tcpEStatsPerfCurSsthresh
MaxSsthresh	tcpEStatsStackMaxSsthresh
MinSsthresh	tcpEStatsStackMinSsthresh

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

TCP_ESTATS SND CONG ROS_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS SND CONG ROS_v0 结构包含针对 TCP 连接的最大拥塞时段的扩展 TCP 统计信息的只读静态信息。

语法

C++

```
typedef struct _TCP_ESTATS_SND_CONG_ROS_v0 {
    ULONG LimCwnd;
} TCP_ESTATS_SND_CONG_ROS_v0, *PTCP_ESTATS_SND_CONG_ROS_v0;
```

成员

LimCwnd

可以使用的拥塞窗口的最大大小（以字节为单位）。

注解

TCP_ESTATS SND CONG ROS_v0 结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS SND CONG ROS_v0 定义为结构版本 0，用于获取有关 TCP 连接的基本发送方拥塞数据的只读动态信息。建立连接后，此信息可用。

在 EstatsType 参数中传递 [TcpConnectionEstatsSndCong](#) 时，可通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索 TCP_ESTATS SND CONG ROS_v0 结构。需要启用扩展 TCP 统计信息才能检索此结构。

TCP 拥塞控制上的 IETF RFC 详细讨论了 TCP 拥塞控制和拥塞控制算法。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc2581.txt>。

此结构的成员在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4898.txt>。

下面是将 TCP_ESTATS_SND_CONG_ROS_v0 结构中的成员映射到 RFC 4898 中为扩展 TCP 统计信息定义的条目：

术语	说明
LimCwnd	tcpEStatsTuneLimCwnd

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

TCP_ESTATS SND CONG RW_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS SND CONG RW_v0 结构包含有关 TCP 连接的发送方拥塞的扩展 TCP 统计信息的读/写配置信息。

语法

C++

```
typedef struct _TCP_ESTATS_SND_CONG_RW_v0 {
    BOOLEAN EnableCollection;
} TCP_ESTATS_SND_CONG_RW_v0, *PTCP_ESTATS_SND_CONG_RW_v0;
```

成员

EnableCollection

一个值，该值指示是否应针对发送方拥塞收集 TCP 连接的扩展统计信息。

如果此成员设置为 TRUE，则会启用 TCP 连接上的扩展统计信息。如果此成员设置为 FALSE，则会禁用 TCP 连接的扩展统计信息。

未设置时此成员的默认状态处于禁用状态。

注解

TCP_ESTATS SND CONG RW_v0 结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS SND CONG RW_v0 定义为结构版本 0，用于提供有关 TCP 连接的发送方拥塞的读/写配置信息。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsSndCongs](#) 时，会使用此结构以及 [SetPerTcp6ConnectionEStats](#) 和 [SetPerTcpConnectionEStats](#) 函数启用和禁用 TCP 连接发送方拥塞的扩展 TCP 统计信息。

在 *EstatsType* 参数中传递 [TcpConnectionEstatsSndCong](#) 时，将通过调用 [GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索

TCP_ESTATS SND CONG RW_v0结构。

要求

最低受支持的客户端	
	Windows Vista [仅限桌面应用]
最低受支持的服务器	
	Windows Server 2008 [仅限桌面应用]
标头	
	tcpstats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

TCP_ESTATS_SYN_OPTS_ROS_v0 结构 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_SYN_OPTS_ROS_v0 结构包含用于 TCP 连接的 SYN 交换上的扩展 TCP 统计信息的只读静态信息。

语法

C++

```
typedef struct _TCP_ESTATS_SYN_OPTS_ROS_v0 {
    BOOLEAN ActiveOpen;
    ULONG    MssRcvd;
    ULONG    MssSent;
} TCP_ESTATS_SYN_OPTS_ROS_v0, *PTCP_ESTATS_SYN_OPTS_ROS_v0;
```

成员

ActiveOpen

类型： BOOLEAN

一个值，该值指示 TCP 连接是否为活动打开状态。

如果本地连接遍历 SYN-SENT 状态，则此成员设置为 TRUE。 否则，此成员设置为 FALSE。

MssRcvd

类型： ULONG

在 SYN 交换期间，在最大段大小 (MSS) 选项中收到的值;如果未收到 MSS 选项，则为零。

此值是远程主机可以接收的单个 TCP 数据报中的最大数据。

MssSent

类型： ULONG

在 SYN 交换期间在 MSS 选项中发送的值;如果未发送 MSS 选项，则为零。

注解

TCP_ESTATS_SYN_OPTS_ROS_v0结构用作 Windows Vista 及更高版本上提供的 TCP 扩展统计信息功能的一部分。

TCP_ESTATS_SYN_OPTS_ROS_v0定义为结构版本 0，用于在 TCP 连接的 SYN 交换中获取只读静态信息。TCP 协议不允许在 SYN 交换后更改此结构的成员。此信息在 SYN 交换完成后可用。

当[TcpConnectionEstatsSynOpts](#) 在 EstatsType 参数中传递时，将通过调用[GetPerTcp6ConnectionEStats](#) 或 [GetPerTcpConnectionEStats](#) 函数来检索 TCP_ESTATS_SYN_OPTS_ROS_v0结构。无需启用扩展 TCP 统计信息来检索此结构。

MssRcvd 和 MSSent 成员中的 MSS 是单个 TCP 数据报中的最大数据。MSS 可以是非常大的值。

此结构的成员在 TCP 扩展统计信息 MIB 上的 IETF RFC 中定义。有关详细信息，请参阅<http://www.ietf.org/rfc/rfc4898.txt>。

下面是将 TCP_ESTATS_SYN_OPTS_ROS_v0 结构中的成员映射到 RFC 4898 中为扩展 TCP 统计信息定义的条目：

术语	说明
ActiveOpen	tcpEStatsStackActiveOpen
MssRcvd	tcpEStatsStackMSSRcvd
MSSent	tcpEStatsStackMSSSent

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

GetPerTcpConnectionEStats

TCP_ESTATS_TYPE

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCPIP_OWNER_MODULE_BASIC_INFO 结构 (iprtrmib.h)

项目2023/08/24

TCPIP_OWNER_MODULE_BASIC_INFO 结构包含指向与 TCP 连接关联的模块名称和模块路径值的指针。 TCPIP_OWNER_MODULE_BASIC_INFO 结构由 [GetOwnerModuleFromTcpEntry](#) 和 [GetOwnerModuleFromTcp6Entry](#) 函数返回。

语法

C++

```
typedef struct _TCPIP_OWNER_MODULE_BASIC_INFO {
    PWCHAR pModuleName;
    PWCHAR pModulePath;
} TCPIP_OWNER_MODULE_BASIC_INFO, *PTCPIP_OWNER_MODULE_BASIC_INFO;
```

成员

pModuleName

指向模块名称的指针。 传递给 [GetOwnerModuleFromTcpEntry](#) 或 [GetOwnerModuleFromTcp6Entry](#) 函数时，此字段应为 NULL 指针。

pModulePath

指向模块的完整路径（包括模块名称）的指针。 传递给 [GetOwnerModuleFromTcpEntry](#) 或 [GetOwnerModuleFromTcp6Entry](#) 函数时，此字段应为 NULL 指针。

注解

如果模块所有者是系统内核，则 IpModuleName 和 IpModulePath 成员指向包含“System”的宽字符串。

在 Windows Vista 及更高版本以及 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，TCPIP_OWNER_MODULE_BASIC_INFO 结构在 *iprtrmib.h* 头文件中定义。

要求

最低受支持的客户端	Windows Vista、Windows XP SP2 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]
标头	iprtrmib.h

反馈

此页面是否有帮助?



[在 Microsoft Q&A 获取帮助](#)

ipexport.h) (TCP_RESERVE_PORT_RANGE 结构)

项目2023/08/24

TCP_RESERVE_PORT_RANGE结构指定要保留的 TCP 端口范围。

语法

C++

```
typedef struct tcp_reserve_port_range {
    USHORT UpperRange;
    USHORT LowerRange;
} TCP_RESERVE_PORT_RANGE, *PTCP_RESERVE_PORT_RANGE;
```

成员

UpperRange

要保留的 TCP 端口范围上限的值。

LowerRange

要保留的 TCP 端口范围下限的值。

要求

 展开表

最低受支持的客户端	Windows 2000 Professional [仅限桌面应用]
最低受支持的服务器	Windows 2000 Server [仅限桌面应用]
标头	ipexport.h

另请参阅

[IP 帮助程序起始页](#)

反馈

此页面是否有帮助?

是

否

IP Helper 枚举

项目 • 2023/06/12

以下枚举与 IP 帮助程序一起使用。

- [DNS_SERVER_PROPERTY_TYPE](#)
- [IF_OPER_STATUS](#)
- [IP_DAD_STATE](#)
- [IP_PREFIX_ORIGIN](#)
- [IP_SUFFIX_ORIGIN](#)
- [NET_ADDRESS_FORMAT](#)
- [NL_NETWORK_CONNECTIVITY_LEVEL_HINT](#)
- [NL_NETWORK_CONNECTIVITY_COST_HINT](#)
- [SCOPE_LEVEL](#)
- [TCP_BOOLEAN_OPTIONAL](#)
- [TCP_ESTATS_TYPE](#)
- [TCP_SOFT_ERROR](#)
- [TCP_TABLE_CLASS](#)
- [TCPIP_OWNER_MODULE_INFO_CLASS](#)
- [UDP_TABLE_CLASS](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获取帮助](#)

IF_OPER_STATUS枚举 (ifdef.h)

项目2023/08/24

IF_OPER_STATUS枚举指定接口的操作状态。

语法

C++

```
typedef enum {
    IfOperStatusUp = 1,
    IfOperStatusDown,
    IfOperStatusTesting,
    IfOperStatusUnknown,
    IfOperStatusDormant,
    IfOperStatusNotPresent,
    IfOperStatusLowerLayerDown
} IF_OPER_STATUS;
```

常量

 展开表

IfOperStatusUp

值: 1

接口已启动并正常运行。 接口能够传递数据包。

IfOperStatusDown

接口未关闭且无法运行。 接口无法传递数据包。

IfOperStatusTesting

接口正在测试中。

IfOperStatusUnknown

接口状态未知。

IfOperStatusDormant

接口不是

用于传递数据包的条件。 接口未启动，但
处于挂起状态，等待某个外部事件。 此状态标识以下情况：
接口正在等待事件将其置于 up 状态。

IfOperStatusNotPresent

此状态是对向下状态的优化
指示接口已关闭，具体是因为
某些组件（例如，硬件组件）不存在
系统。

IfOperStatusLowerLayerDown

此状态是向下状态的优化。
接口可操作，但接口下方的网络层无法正常运行。

注解

IF_OPER_STATUS 枚举在 [IP_ADAPTER_ADDRESSES](#) 结构的 OperStatus 成员中使用。

要求

[+] 展开表

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	ifdef.h

另请参阅

[IP_ADAPTER_ADDRESSES](#)

反馈

此页面是否有帮助？

是

否

NL_DAD_STATE枚举 (nldef.h)

项目2023/08/25

IP_DAD_STATE 枚举指定有关 IPv4 或 IPv6 地址的重复地址检测 (DAD) 状态的信息。

语法

C++

```
typedef enum {
    NldsInvalid,
    NldsTentative,
    NldsDuplicate,
    NldsDeprecated,
    NldsPreferred,
    IpDadStateInvalid = 0,
    IpDadStateTentative,
    IpDadStateDuplicate,
    IpDadStateDeprecated,
    IpDadStatePreferred
} NL_DAD_STATE;
```

常量

[+] 展开表

NldsInvalid

NldsTentative

NldsDuplicate

NldsDeprecated

NldsPreferred

IpDadStateInvalid

值: 0

DAD 状态无效。

IpDadStateTentative

DAD 状态为暂定状态。

`IpDadStateDuplicate`

检测到重复的 IP 地址。

`IpDadStateDeprecated`

IP 地址已弃用。

`IpDadStatePreferred`

IP 地址是首选地址。

注解

`IP_DAD_STATE` 枚举在 [IP_ADAPTER_UNICAST_ADDRESS](#) 结构的 `DadState` 成员中使用。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织方式已更改，`IP_DAD_STATE` 枚举在 *Iptypes.h* 头文件自动包含的 *Nldef.h* 头文件中定义。永远不应直接使用 *Nldef.h* 和 *Iptypes.h* 头文件。

要求

 展开表

最低受支持的客户端 Windows XP [仅限桌面应用]

最低受支持的服务器 Windows Server 2003 [仅限桌面应用]

标头 `nldef.h` (包括 Windows 8 上的 `Iphlpapi.h`、Windows Server 2008 R2、Windows 7、Windows Server 2008 Windows Vista)

请参阅

[IP_ADAPTER_UNICAST_ADDRESS](#)

反馈

此页面是否有帮助？

 是

 否

NL_PREFIX_ORIGIN枚举 (nldef.h)

项目2023/08/25

IP_PREFIX_ORIGIN枚举指定 IPv4 或 IPv6 地址前缀的来源，并与 IP_ADAPTER_UNICAST_ADDRESS 结构一起使用。

语法

C++

```
typedef enum {
    IpPrefixOriginOther = 0,
    IpPrefixOriginManual,
    IpPrefixOriginWellKnown,
    IpPrefixOriginDhcp,
    IpPrefixOriginRouterAdvertisement,
    IpPrefixOriginUnchanged = 1 << 4
} NL_PREFIX_ORIGIN;
```

常量

[] 展开表

`IpPrefixOriginOther`

值: 0

IP 前缀由此枚举中定义的源以外的其他源提供。

`IpPrefixOriginManual`

IP 地址前缀是手动指定的。

`IpPrefixOriginWellKnown`

IP 地址前缀来自已知源。

`IpPrefixOriginDhcp`

IP 地址前缀由 DHCP 设置提供。

`IpPrefixOriginRouterAdvertisement`

IP 地址前缀是通过路由器广播 (RA) 获取的。

`IpPrefixOriginUnchanged`

值: 1

IP 地址前缀应保持不变。当 IP 前缀源的值应保持不变时，在设置单播 IP 接口的属性时，将使用此值。

注意 此枚举值仅在 Windows Vista 及更高版本上可用。

注解

IP_PREFIX_ORIGIN 枚举用于 [IP_ADAPTER_UNICAST_ADDRESS](#) 结构的 PrefixOrigin 成员。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织方式已更改，IP_PREFIX_ORIGIN 枚举在 *Iptypes.h* 头文件自动包含的 *Nldef.h* 头文件中定义。若要使用 IP_PREFIX_ORIGIN 枚举，必须在 *Iptypes.h* 头文件之前包含 *Winsock2.h* 头文件。

要求

 展开表

最低受支持的客户端 Windows XP [仅限桌面应用]

最低受支持的服务器 Windows Server 2003 [仅限桌面应用]

标头 *nldef.h* (包括 Windows 8 上的 *Iphlpapi.h*、Windows Server 2008 R2、Windows 7、Windows Server 2008 Windows Vista)

请参阅

[IP_ADAPTER_UNICAST_ADDRESS](#)

反馈

此页面是否有帮助?

是

否

NL_SUFFIX_ORIGIN 枚举 (nldef.h)

项目2023/08/25

[IP_SUFFIX_ORIGIN 枚举指定 IPv4 或 IPv6 地址后缀的原点，并与 IP_ADAPTER_UNICAST_ADDRESS 结构一起使用。](#)

语法

C++

```
typedef enum {
    NlsoOther = 0,
    NlsoManual,
    NlsoWellKnown,
    NlsoDhcp,
    NlsoLinkLayerAddress,
    NlsoRandom,
    IpSuffixOriginOther = 0,
    IpSuffixOriginManual,
    IpSuffixOriginWellKnown,
    IpSuffixOriginDhcp,
    IpSuffixOriginLinkLayerAddress,
    IpSuffixOriginRandom,
    IpSuffixOriginUnchanged = 1 << 4
} NL_SUFFIX_ORIGIN;
```

常量

[展开表](#)

NlsoOther

值: 0

NlsoManual

NlsoWellKnown

NlsoDhcp

NlsoLinkLayerAddress

NlsoRandom

<code>IpSuffixOriginOther</code>
值: 0
IP 地址后缀由此枚举中定义的源以外的其他源提供。
<code>IpSuffixOriginManual</code>
IP 地址后缀是手动指定的。
<code>IpSuffixOriginWellKnown</code>
IP 地址后缀来自已知源。
<code>IpSuffixOriginDhcp</code>
IP 地址后缀由 DHCP 设置提供。
<code>IpSuffixOriginLinkLayerAddress</code>
IP 地址后缀是从链接层地址获取的。
<code>IpSuffixOriginRandom</code>
IP 地址后缀是从随机源获取的。
<code>IpSuffixOriginUnchanged</code>
值: 1
IP 地址后缀应保持不变。当 IP 后缀原点的值应保持不变时，设置单播 IP 接口的属性时，将使用此值。

注意 此枚举值仅适用于 Windows Vista 及更高版本。

注解

`IP_SUFFIX_ORIGIN` 枚举在 `IP_ADAPTER_UNICAST_ADDRESS` 结构的 `PrefixOrigin` 成员中使用。

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，`IP_SUFFIX_ORIGIN` 枚举在 `Iptypes.h` 头文件自动包含的 `Nldef.h` 头文件中定义。若要使用 `IP_SUFFIX_ORIGIN` 枚举，必须在 `Iptypes.h` 头文件之前包含 `Winsock2.h` 头文件。

要求

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	nldef.h (包括 Windows 8、Windows Server 2008 R2、Windows 7、Windows Server 2008 Windows Vista)

请参阅

[IP_ADAPTER_UNICAST_ADDRESS](#)

反馈

此页面是否有帮助?

 是

 否

NET_ADDRESS_FORMAT 枚举 (iphlpapi.h)

项目2023/08/24

NET_ADDRESS_FORMAT 枚举指定 [ParseNetworkString](#) 函数返回的网络地址的格式。

语法

C++

```
typedef enum NET_ADDRESS_FORMAT_ {
    NET_ADDRESS_FORMAT_UNSPECIFIED = 0,
    NET_ADDRESS_DNS_NAME,
    NET_ADDRESS_IPV4,
    NET_ADDRESS_IPV6
} NET_ADDRESS_FORMAT;
```

常量

NET_ADDRESS_FORMAT_UNSPECIFIED

值: 0

未指定网络地址的格式。

NET_ADDRESS_DNS_NAME

网络地址的格式为 DNS 名称。

NET_ADDRESS_IPV4

网络地址的格式是 IPv4 的 Internet 标准点十进制表示法中的字符串。

NET_ADDRESS_IPV6

网络地址的格式是 IPv6 的 Internet 标准十六进制编码字符串。

注解

NET_ADDRESS_FORMAT 枚举在 Windows Vista 及更高版本上定义。

[ParseNetworkString](#) 函数返回的 [NET_ADDRESS_INFO](#) 结构。

要求

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	iphlpapi.h

另请参阅

[NET_ADDRESS_INFO](#)

[ParseNetworkString](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)

SCOPE_LEVEL枚举 (ws2def.h)

项目2023/08/28

SCOPE_LEVEL 枚举与 [IP_ADAPTER_ADDRESSES](#) 结构一起使用，以标识 IPv6 地址的范围级别。

语法

C++

```
typedef enum {
    ScopeLevelInterface = 1,
    ScopeLevelLink = 2,
    ScopeLevelSubnet = 3,
    ScopeLevelAdmin = 4,
    ScopeLevelSite = 5,
    ScopeLevelOrganization = 8,
    ScopeLevelGlobal = 14,
    ScopeLevelCount = 16
} SCOPE_LEVEL;
```

常量

 展开表

`ScopeLevelInterface`

值: 1

范围为接口级别。

`ScopeLevelLink`

值: 2

范围为链接级别。

`ScopeLevelSubnet`

值: 3

范围为子网级别。

`ScopeLevelAdmin`

值: 4

范围为管理级别。

`ScopeLevelSite`

值: 5

范围为站点级别。

`ScopeLevelOrganization`

值: 8

范围为组织级别。

`ScopeLevelGlobal`

值: 14

范围为全局。

`ScopeLevelCount`

值: 16

注解

`SCOPE_LEVEL` 枚举在 [IP_ADAPTER_ADDRESSES](#) 结构的 `ZoneIndices` 成员中使用。

在 Windows Vista 和更高版本以及 Microsoft Windows 软件开发工具包 (SDK) 上, 头文件的组织已更改, `SCOPE_LEVEL` 枚举类型在 `Ws2def.h` 头文件中定义。请注意, `Ws2def.h` 头文件会自动包含在 `Winsock2.h` 中, 永远不应直接使用。

要求

[+] 展开表

最低受支持的客户端	Windows XP [仅限桌面应用]
最低受支持的服务器	Windows Server 2003 [仅限桌面应用]
标头	<code>ws2def.h</code> (包括 <code>Winsock2.h</code>)

另请参阅

[IP_ADAPTER_ADDRESSES](#)

反馈

此页面是否有帮助?

是

否

TCP_BOOLEAN_OPTIONAL 枚举 (tcppestats.h)

项目2023/08/26

TCP_BOOLEAN_OPTIONAL 枚举定义调用方在更新 TCP 连接的读/写信息中的成员时可以指定的状态。

语法

C++

```
typedef enum _TCP_BOOLEAN_OPTIONAL {
    TcpBoolOptDisabled = 0,
    TcpBoolOptEnabled,
    TcpBoolOptUnchanged = -1
} TCP_BOOLEAN_OPTIONAL, *PTCP_BOOLEAN_OPTIONAL;
```

常量

TcpBoolOptDisabled

值: 0

应禁用 选项。

TcpBoolOptEnabled

应启用 选项。

TcpBoolOptUnchanged

值: -1

选项应保持不变。

注解

TCP_BOOLEAN_OPTIONAL 枚举在 Windows Vista 及更高版本上定义。

使用对 [SetPerTcp6ConnectionEStats](#) 和 [SetPerTcpConnectionEStats](#) 函数的调用启用和禁用 TCP 连接上的扩展统计信息集合，其中指定的扩展统计信息类型是 [TCP_ESTATS_TYPE](#) 枚举类型中的值之一。TCP_BOOLEAN_OPTIONAL 枚举中的值用于指定应如何更新 [TCP_ESTATS_BANDWIDTH_RW_v0](#) 结构中的成员，以启用或禁用 TCP 连接上的扩展统计信息，以便估计带宽。

要求

最低受支持的客户端	
	Windows Vista [仅限桌面应用]
最低受支持的服务器	
	Windows Server 2008 [仅限桌面应用]
标头	
	tcpstats.h

另请参阅

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_BANDWIDTH_RW_v0](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

[在 Microsoft Q&A 获得帮助](#)

TCP_ESTATS_TYPE 枚举 (tcppestats.h)

项目2023/08/26

TCP_ESTATS_TYPE枚举定义请求或正在设置的 TCP 连接的扩展统计信息的类型。

语法

C++

```
typedef enum {
    TcpConnectionEstatsSynOpts,
    TcpConnectionEstatsData,
    TcpConnectionEstatsSndCong,
    TcpConnectionEstatsPath,
    TcpConnectionEstatsSendBuff,
    TcpConnectionEstatsRec,
    TcpConnectionEstatsObsRec,
    TcpConnectionEstatsBandwidth,
    TcpConnectionEstatsFineRtt,
    TcpConnectionEstatsMaximum
} TCP_ESTATS_TYPE, *PTCP_ESTATS_TYPE;
```

常量

[+] 展开表

TcpConnectionEstatsSynOpts

此值指定 TCP 连接的 SYN 交换信息。

此枚举值仅提供只读静态信息。

TcpConnectionEstatsData

此值指定 TCP 连接的扩展数据传输信息。

此枚举值只能使用只读动态信息和读/写信息。

TcpConnectionEstatsSndCong

此值指定 TCP 连接的发送方拥塞。

) (只读静态、只读动态和读/写信息的所有三种类型的信息都可用于此枚举值。

TcpConnectionEstatsPath

此值指定 TCP 连接的扩展路径度量信息。 此信息用于推断段

对从本地发送方到远程的路径重新排序接收机。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsSendBuff`

此值指定 TCP 连接的扩展输出队列信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsRec`

此值指定 TCP 连接的扩展本地接收器信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsObsRec`

此值指定 TCP 连接的扩展远程接收器信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsBandwidth`

此值指定带宽上的 TCP 连接的带宽估计统计信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsFineRtt`

此值指定 TCP 连接的精细往返时间 (RTT) 估计统计信息。

此枚举值只能使用只读动态信息和读/写信息。

`TcpConnectionEStatsMaximum`

`TCP_ESTATS_TYPE_STATE` 枚举类型的最大值。对于 TCP 连接的可能类型的扩展统计信息，这不是法律值。

注解

`TCP_ESTATS_TYPE` 枚举在 Windows Vista 及更高版本上定义。

`GetPerTcp6ConnectionEStats` 和 `GetPerTcp6ConnectionEStats` 函数旨在使用 TCP 诊断网络和应用程序中的性能问题。如果基于网络的应用程序性能不佳，TCP 可以确定瓶颈是在发送方、接收方还是网络本身。如果瓶颈在网络中，TCP 可以提供有关其性质的特定信息。

`GetPerTcp6ConnectionEStats` 和 `GetPerTcp6ConnectionEStats` 函数用于根据使用 `TCP_ESTATS_TYPE` 枚举类型中的值之一指定的扩展统计信息的类型检索 TCP 连接的扩展统计信息。使用对 `SetPerTcp6ConnectionEStats` 和 `SetPerTcpConnectionEStats` 函数的调

用启用和禁用 TCP 连接上的扩展统计信息集合，其中指定的扩展统计信息的类型是 **TCP_ESTATS_TYPE** 枚举类型中的值之一。

要求

 展开表

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcppestats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[SetPerTcp6ConnectionEStats](#)

[SetPerTcpConnectionEStats](#)

[TCP_ESTATS_BANDWIDTH_ROD_v0](#)

[TCP_ESTATS_BANDWIDTH_RW_v0](#)

[TCP_ESTATS_DATA_ROD_v0](#)

[TCP_ESTATS_DATA_RW_v0](#)

[TCP_ESTATS_FINE_RTT_ROD_v0](#)

[TCP_ESTATS_FINE_RTT_RW_v0](#)

[TCP_ESTATS_OBS_REC_ROD_v0](#)

[TCP_ESTATS_OBS_REC_RW_v0](#)

[TCP_ESTATS_PATH_ROD_v0](#)

[TCP_ESTATS_PATH_RW_v0](#)

[TCP_ESTATS_REC_ROD_v0](#)

TCP_ESTATS_REC_RW_v0

TCP_ESTATS_SEND_BUFF_ROD_v0

TCP_ESTATS_SEND_BUFF_RW_v0

TCP_ESTATS SND CONG ROD_v0

TCP_ESTATS SND CONG ROS_v0

TCP_ESTATS SND CONG RW_v0

TCP_ESTATS_SYN_OPTS_ROS_v0

反馈

此页面是否有帮助?

 是

 否

TCP_SOFT_ERROR 枚举 (tcppestats.h)

项目2023/08/26

TCP_SOFT_ERROR枚举定义 TCP 连接上记录的非致命或软错误的原因。

语法

C++

```
typedef enum {
    TcpErrorNone = 0,
    TcpErrorBelowDataWindow,
    TcpErrorAboveDataWindow,
    TcpErrorBelowAckWindow,
    TcpErrorAboveAckWindow,
    TcpErrorBelowTsWindow,
    TcpErrorAboveTsWindow,
    TcpErrorDataChecksumError,
    TcpErrorDataLengthError,
    TcpErrorMaxSoftError
} TCP_SOFT_ERROR, *PTCP_SOFT_ERROR;
```

常量

[+] 展开表

TcpErrorNone

值: 0

未发生软错误。

TcpErrorBelowDataWindow

段中的所有数据都如下所示

发送未确认 (SND。 UNA) 序列号。对于保持连接和零窗口探测，此软错误是正常的。

TcpErrorAboveDataWindow

段中的某些数据位于上面

发送窗口 (SND。 WND) 大小。此软错误表示存在实现 bug 或可能攻击。

TcpErrorBelowAckWindow

在 SND 下方收到 ACK。UNA 序列号。此软错误指示返回路径正在对 ACK 重新排序。

TcpErrorAboveAckWindow

对于我们尚未发送的数据，已收到 ACK。
此软错误表示存在实现 bug 或可能的攻击。

TcpErrorBelowTsWindow

段上的时间戳回显回复 (TSecr) 早于
当前 TS。最近 (时间戳，每当
段) 发送。此错误适用于使用由 RFC 1323 中的 IETF 定义的 TCP 时间戳选项 (TSopt) 的 TCP 连接。
有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc1323.txt>。此软错误对于保护防止包装的极少数情况是正常的
PAWS (序列号
机制检测网络重新排序的数据。

TcpErrorAboveTsWindow

段上的 TSecr 比
当前 TS。最近。此软错误表示实现 bug 或
可能的攻击。

TcpErrorDataChecksumError

收到错误的 TCP 校验和。请注意，此值
在本质上是脆弱的，因为标头字段用于
标识连接可能已损坏。

TcpErrorDataLengthError

发生数据长度错误。

此值未在 TCP 扩展统计信息 MIB 上的 IETF 草稿 RFC 中定义。

TcpErrorMaxSoftError

`TCP_SOFT_ERROR_STATE` 枚举类型的可能最大值。由于 TCP 连接出现软错误的原因，这不是法定值。

注解

`TCP_SOFT_ERROR` 枚举在 Windows Vista 及更高版本上定义。

此枚举中的值在 TCP 扩展统计信息 MIB 上的 IETF 草稿 RFC 中定义。有关详细信息，请参阅 <http://www.ietf.org/rfc/rfc4898.txt>。

要求

 展开表

最低受支持的客户端	Windows Vista [仅限桌面应用]
最低受支持的服务器	Windows Server 2008 [仅限桌面应用]
标头	tcpstats.h

另请参阅

[GetPerTcp6ConnectionEStats](#)

[GetPerTcpConnectionEStats](#)

[TCP_ESTATS_TYPE](#)

反馈

此页面是否有帮助?

 是

 否

TCP_TABLE_CLASS 枚举 (iprtrmib.h)

项目2023/08/24

TCP_TABLE_CLASS枚举定义用于指示调用 [GetExtendedTcpTable](#) 返回的表类型的值集。

语法

C++

```
typedef enum _TCP_TABLE_CLASS {
    TCP_TABLE_BASIC_LISTENER,
    TCP_TABLE_BASIC_CONNECTIONS,
    TCP_TABLE_BASIC_ALL,
    TCP_TABLE_OWNER_PID_LISTENER,
    TCP_TABLE_OWNER_PID_CONNECTIONS,
    TCP_TABLE_OWNER_PID_ALL,
    TCP_TABLE_OWNER_MODULE_LISTENER,
    TCP_TABLE_OWNER_MODULE_CONNECTIONS,
    TCP_TABLE_OWNER_MODULE_ALL
} TCP_TABLE_CLASS, *PTCP_TABLE_CLASS;
```

常量

`TCP_TABLE_BASIC_LISTENER`

将返回一个[MIB_TCPTABLE](#) 表，其中包含仅接收本地计算机上的 TCP 终结点) 所有侦听 (。

`TCP_TABLE_BASIC_CONNECTIONS`

包含本地计算机上所有连接的 TCP 终结点的 [MIB_TCPTABLE](#) 表将返回到调用方。

`TCP_TABLE_BASIC_ALL`

包含本地计算机上所有 TCP 终结点的 [MIB_TCPTABLE](#) 表将返回到调用方。

`TCP_TABLE_OWNER_PID_LISTENER`

包含仅接收本地计算机上的 TCP 终结点) 所有侦听 (的[MIB_TCPTABLE_OWNER_PID](#)或[MIB_TCP6TABLE_OWNER_PID](#)将返回到调用方。

`TCP_TABLE_OWNER_PID_CONNECTIONS`

[MIB_TCPTABLE_OWNER_PID](#)或[MIB_TCP6TABLE_OWNER_PID](#)将包含本地计算机上所有连接的 TCP 终结点的结构返回到调用方。

`TCP_TABLE_OWNER_PID_ALL`

包含本地计算机上的所有 TCP 终结点的[MIB_TCPTABLE_OWNER_PID](#)或[MIB_TCP6TABLE_OWNER_PID](#)结构将返回到调用方。

TCP_TABLE_OWNER_MODULE_LISTENER

将 [返回给](#) 调用方 MIB_TCPTABLE_OWNER_MODULE 或 MIB_TCP6TABLE_OWNER_MODULE 结构，该结构包含仅接收本地计算机上的 TCP 终结点) 所有侦听(。

TCP_TABLE_OWNER_MODULE_CONNECTIONS

包含本地计算机上所有连接的 TCP 终结点 [的 MIB_TCPTABLE_OWNER_MODULE 或 MIB_TCP6TABLE_OWNER_MODULE](#) 结构将返回到调用方。

TCP_TABLE_OWNER_MODULE_ALL

包含本地计算机上的所有 TCP 终结点[的 MIB_TCPTABLE_OWNER_MODULE 或 MIB_TCP6TABLE_OWNER_MODULE](#)结构将返回到调用方。

注解

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 中，头文件的组织已更改，TCP_TABLE_CLASS 枚举在 *Iprtrmib.h* 头文件中定义，而不是在 *Iphlpapi.h* 头文件中定义。请注意，*Iprtrmib.h* 头文件会自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Iprtrmib.h* 头文件。

要求

最低受支持的客户端 Windows Vista、Windows XP 和 SP2 [仅限桌面应用]

最低受支持的服务器 Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]

标头 *iprtmb.h* (包括 *Iphlpapi.h*)

另请参阅

[GetExtendedTcpTable](#)

反馈

此页面是否有帮助？

是

否

[在 Microsoft Q&A 获取帮助](#)

TCPIP_OWNER_MODULE_INFO_CLASS 枚举 (iprtrmib.h)

项目2023/08/24

TCPIP_OWNER_MODULE_INFO_CLASS 枚举定义传递给
GetOwnerModuleFromXXXEntry 系列调用的模块信息结构的类型。

语法

C++

```
typedef enum _TCPIP_OWNER_MODULE_INFO_CLASS {
    TCPIP_OWNER_MODULE_INFO_BASIC
} TCPIP_OWNER_MODULE_INFO_CLASS, *PTCPIP_OWNER_MODULE_INFO_CLASS;
```

常量

TCPIP_OWNER_MODULE_INFO_BASIC

TCPIP_OWNER_MODULE_BASIC_INFO 结构传递给 GetOwnerModuleFromXXXEntry 函数。

要求

最低受支持的客户端 Windows Vista、Windows XP SP2 [仅限桌面应用]

最低受支持的服务器 Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]

标头 iprtrmib.h

反馈

此页面是否有帮助？

是

否

在 Microsoft Q&A 获取帮助

UDP_TABLE_CLASS 枚举 (iprtrmib.h)

项目2023/08/24

UDP_TABLE_CLASS 枚举定义用于指示调用 [GetExtendedUdpTable](#) 返回的表类型的值集。

语法

C++

```
typedef enum _UDP_TABLE_CLASS {
    UDP_TABLE_BASIC,
    UDP_TABLE_OWNER_PID,
    UDP_TABLE_OWNER_MODULE
} UDP_TABLE_CLASS, *PUDP_TABLE_CLASS;
```

常量

UDP_TABLE_BASIC

包含本地计算机上所有 UDP 终结点的 [MIB_UDPTABLE](#) 结构将返回到调用方。

UDP_TABLE_OWNER_PID

包含本地计算机上所有 UDP 终结点的 [MIB_UDPTABLE_OWNER_PID](#) 或 [MIB_UDP6TABLE_OWNER_PID](#) 结构将返回到调用方。

UDP_TABLE_OWNER_MODULE

包含本地计算机上所有 UDP 终结点的 [MIB_UDPTABLE_OWNER_MODULE](#) 或 [MIB_UDP6TABLE_OWNER_MODULE](#) 结构将返回到调用方。

注解

在为 Windows Vista 及更高版本发布的 Microsoft Windows 软件开发工具包 (SDK) 上，头文件的组织已更改，**UDP_TABLE_CLASS** 枚举在 *Iprtrmib.h* 头文件中定义，而不是在 *Iphlpapi.h* 头文件中定义。请注意，*Iprtrmib.h* 头文件会自动包含在 *Iphlpapi.h* 头文件中。永远不应直接使用 *Iprtrmib.h* 头文件。

要求

最低受支持的客户端	Windows Vista、Windows XP 和 SP2 [仅限桌面应用]
最低受支持的服务器	Windows Server 2008、Windows Server 2003 SP1 [仅限桌面应用]
标头	iprtrmib.h (包括 Iphlpapi.h)

另请参阅

[GetExtendedUdpTable](#)

反馈

此页面是否有帮助？

 是

 否

[在 Microsoft Q&A 获得帮助](#)