Part of the five-volume
Networking Services Developer's Reference Library

Microsoft®

The essential reference set for developing with Microsoft® Windows® networking technologies

David Iseminger
Series Editor
www.iseminger.com

Remote Access Services

BASED ON
msdn™ library

Microsoft®

David Iseminger
Series Editor

Remote Access Services

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
Iseminger, David, 1969-
Networking Services Developer's Reference Library / David Iseminger.
p. cm.
ISBN 0-7356-0993-4
1. Application Software--Development. 2. Microsoft Windows (Computer file). 3. Computer networks. I. Title.
QA76.76.A65 I84 2000
005.4'4769--dc21 00-020241

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 WCWC 5 4 3 2 1 0

Distributed in Canada by Penguin Books Canada Limited.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at mspress.microsoft.com.

Acquisitions Editor: Ben Ryan
Project Editor: Wendy Zucker

Part No. 097-0002786

Acknowledgements

First, thanks to **Ben Ryan** at Microsoft Press for continuing to share my enthusiasm about the series. Many thanks to Ben and **Steve Guty** for also managing the business details associated with publishing this series. We're just getting started!

Wendy Zucker again kept step with the difficult and tight schedule at Microsoft Press and orchestrated things in the way only project editors can endure. **John Pierce** was also instrumental in seeing the publishing process through completion, many thanks to both of them. The cool cover art that will continue through the series is directed by **Greg Hickman**—thanks for the excellent work. I'm a firm believer that artwork and packaging are integral to the success of a project.

Thanks also to the marketing team at Microsoft Press that handles this series: **Cora McLaughlin** and **Cheri Chapman** on the front lines and **Jocelyn Paul** each deserve recognition for their coordination efforts with MSDN, openness to my ideas and suggestions, creative marketing efforts, and other feats of marketing ingenuity.

On the Windows SDK side of things, thanks again to **Morgan Seeley** for introducing me to the editor at Microsoft Press, and thereby routing this series to the right place.

Thanks also to **Margot (Maley) Hutchison** for doing all those agent-ish things so well.

Author's Note In Part 2 you'll see some code blocks that have unusual margin settings, or code that wraps to a subsequent line. This is a result of physical page constraints of printed material; the original code in these places was indented too much to keep its printed form on one line. I've reviewed every line of code in this library in an effort to ensure it reads as well as possible (for example, modifying comments to keep them on one line, and to keep line-delimited comment integrity). In some places, however, the word wrap effect couldn't be avoided. As such, please ensure that you check closely if you use and compile these examples.

Contents

Part 3

CHAPTER 1

Getting Around in the Networking Services Library

Networking is pervasive in this digital age in which we live. Information at your fingertips, distributed computing, name resolution, and indeed the entire Internet—the advent of which will be ascribed to our generation for centuries to come—imply and require networking. Everything that has become the buzz of our business and personal lives, including e-mail, cell phones, and Web surfing, is enabled by the fact that networking has been brought to the masses (and we've barely scraped the beginning of the trend). You, the network-enabled Windows application developer, need to know how to lasso this all-important networking services capability and make it a part of your application. You've come to the right place.

Networking isn't magic, but it can seem that way to those who aren't accustomed to it (or to the programmer who isn't familiar with the technologies or doesn't know how to make networking part of his or her application). That's why the *Networking Services Developer's Reference Library* isn't just a collection of programmatic reference information; it would be only half-complete if it were. Instead, the Networking Services Library is a collection of explanatory and reference information that combine to provide you with the complete set that you need to create today's network-enabled Windows application.

The Networking Services Library is *the* comprehensive reference guide to network-enabled application development. This library, like all libraries in the Windows Programming Reference Series (WPRS), is designed to deliver the most complete, authoritative, and accessible reference information available on a given subject of Windows network programming—without sacrificing focus. Each book in each library is dedicated to a logical group of technologies or development concerns; this approach has been taken specifically to enable you to find the information you need quickly, efficiently, and intuitively.

In addition to its networking services development information, the Networking Services Library contains tips designed to make your programming life easier. For example, a thorough explanation and detailed tour of MSDN Online is included, as is a section that helps you get the most out of your MSDN subscription. Just in case you don't have an MSDN subscription, or don't know why you should, I've included information about that too, including the differences between the three levels of MSDN subscription, what each level offers, and why you'd want a subscription when MSDN Online is available over the Internet.

To ensure that you don't get lost in all the information provided in the Networking Services Library, each volume's appendixes provide an all-encompassing programming directory to help you easily find the particular programming element you're looking for. This directory suite, which covers all the functions, structures, enumerations, and other programming elements found in network-enabled application development, gets you quickly to the volume and page you need, saving you hours of time and bucketsful of frustration.

How the Networking Services Library Is Structured

The Networking Services Library consists of five volumes, each of which focuses on a particular aspect of network programming. These programming reference volumes have been divided into the following:

- Volume 1: Winsock and QOS
- Volume 2: Network Interfaces and Protocols
- Volume 3: RPC and WNet
- Volume 4: Remote Access Services
- Volume 5: Routing

Dividing the Networking Services Library into these categories enables you to quickly identify the Networking Services volume you need, based on your task, and facilitates your maintenance of focus for that task. This approach enables you to keep one reference book open and handy, or tucked under your arm while researching that aspect of Windows programming on sandy beaches, without risking back problems (from toting around all 3,000+ pages of the Networking Services Library) and without having to shuffle among multiple less-focused books.

Within the Networking Services Library—and in fact, in all WPRS Libraries—each volume has a deliberate structure. This per-volume structure has been created to further focus the reference material in a developer-friendly manner, to maintain consistency within each volume and each Library throughout the series, and to enable you to easily gather the information you need. To that end, each volume in the Networking Services Library contains the following parts:

- Part 1: Introduction and Overview
- Part 2: Guides, Examples, and Programmatic Reference
- Part 3: Intelligently Structured Indexes

Part 1 provides an introduction to the Networking Services Library and to the WPRS (what you're reading now), and a handful of chapters designed to help you get the most out of networking technologies, MSDN, and MSDN Online. MSDN and WPRS Libraries are your tools in the developer process; knowing how to use them to their fullest will enable you to be more efficient and effective (both of which are generally desirable traits). In certain volumes (where appropriate), I've also provided additional information that you'll need in your network-enabled development efforts, and included such information as concluding chapters in Part 1. For example, Volume 3 includes a chapter that explains terms used throughout the RPC development documentation; by putting it into Chapter 5 of that volume, you always know where to go when you have a question about an RPC term. Some of the other volumes in the Networking Services Library conclude their Part 1 with chapters that include information crucial to their volume's contents, but I've been very selective about including such information. Publishing constraints have limited the amount of information I can provide in each volume (and in the library as a whole), so I've focused on the priority: getting you the most useful information possible within the number of pages I have to work with.

Part 2 contains the networking reference material particular to its volume. You'll notice that each volume contains *much* more than simple collections of function and structure definitions. A comprehensive reference resource should include information about how to use a particular technology, as well as definitions of programming elements. Consequently, the information in Part 2 combines complete programming element definitions with instructional and explanatory material for each programming area.

Part 3 is a collection of intelligently arranged and created indexes. One of the biggest challenges of the IT professional is finding information in the sea of available resources and network programming is probably one of the most complex and involved of any development discipline. In order to help you get a handle on network programming references (and Microsoft technologies in general), Part 3 puts all such information into an understandable, manageable directory (in the form of indexes) that enables you to quickly find the information you need.

How the Networking Services Library Is Designed

The Networking Services Library (and all libraries in the WPRS) is designed to deliver the most pertinent information in the most accessible way possible. The Networking Services Library is also designed to integrate seamlessly with MSDN and MSDN Online by providing a look and feel consistent with their electronic means of disseminating Microsoft reference information. In other words, the way a given function reference appears on the pages of this book has been designed specifically to emulate the way that MSDN and MSDN Online present their function reference pages.

The reason for maintaining such integration is simple: to make it easy for you to use the tools and get the ongoing information you need to create quality programs. Providing a "common interface" among reference resources allows your familiarity with the Networking Services Library reference material to be immediately applied to MSDN or MSDN Online, and vice-versa. In a word, it means *consistency*.

You'll find this philosophy of consistency and simplicity applied throughout WPRS publications. I've designed the series to go hand-in-hand with MSDN and MSDN Online resources. Such consistency lets you leverage your familiarity with electronic reference material, then apply that familiarity to enable you to get away from your computer if you'd like, take a book with you, and—in the absence of keyboards and e-mail and upright chairs—get your programming reading and research done. Of course, each of the Networking Services Library volumes fits nicely right next to your mouse pad as well, even when opened to a particular reference page.

With any job, the simpler and more consistent your tools are, the more time you can spend doing work rather than figuring out how to use your tools. The structure and design of the Networking Services Library provide you with a comprehensive, presharpened toolset to build compelling Windows applications.

CHAPTER 2

What's In This Volume?

Volume 4 of the *Networking Services Developer's Reference Library* gives its undivided attention to Remote Access Services, commonly referred to simply as RAS.

The Remote Access Service (RAS) API is included in Microsoft Windows NT 4.0. RAS is used to create client applications that can display any of the Routing and RAS common dialog boxes, start and end a remote access connection, manipulate phone-book entries and network addresses that are mapped to phone-book entries, and get information about existing RAS connection status or RAS-capable devices.

RAS makes it possible to connect a remote client computer to a network server over a Wide Area Network (WAN) link or a Virtual Private Network (VPN). The remote computer can then participate on the server's LAN as though the remote computer was connected to the LAN directly. The RAS API enables programmers to access the features of RAS programmatically. The API is applicable in any networking environment that utilizes RAS. Part 2 of this volume provides a complete treatment of RAS.

This volume also has information about how you can use development resources such as MSDN, MSDN Online, and developer support resources. This helpful information is found in various chapters in Part 1, and those chapters are common to all WPRS volumes. By including this information in each library and in each volume, a few goals of the WPRS are achieved:

- I don't presume you have bought, or expect you to have to buy another WPRS Library to get access to this information. Maybe your primary focus is network programming, and your budget doesn't allow for you to purchase the *Active Directory Developer's Reference Library*. Since I've included this information in this library, you don't have to.
- You can access this important and useful information regardless of which volume you have in your hand. You don't have to (nor *should* you have to) fumble with another physical book to refer to information about how to get the most out of MSDN, or where to get support for questions you have about a particular Windows development problem you're having.
- Each volume becomes more useful, more portable, and more complete in and of itself. This goal of the WPRS makes it easier for you to grab one of its libraries' volumes and take it with you, rather than feeling like you must bring multiple volumes with you to have access to the library's important overview and usability information.

These goals have steered this library's content and choices of included technologies; I hope you find its information is useful, portable, a good value, and as accessible as it can be.

Part 2 of this volume provides RAS information in the following chapter-based focuses:

RAS Programming Guide

This guide takes you through the steps necessary to implement RAS capabilities in your Windows application. All such tasks are grouped in task-oriented categories, such as connection operations, AutoDial, server administration, and more.

RAS Reference

A collection of chapters appears after the RAS programming guide that provide a complete treatment of the RAS API.

RRAS Overview

This chapter provides an overview of the new Remote Access capabilities built into RRAS, which is the successor of RAS.

RAS Administration

This chapter provides information and programmatic reference for performing RAS Administration programming using RRAS-based RAS administration. Where there are differences in the treatment of RAS on Windows NT 4.0 and Windows 2000, such differences are clearly noted in the text.

EAP

Windows 2000 supports the Extensible Authentication Protocol (EAP). EAP allows third-party authentication modules to interact with the implementation of the Point-to-Point Protocol (PPP) included in Windows 2000 Remote Access Service (RAS).

EAP is an extension to PPP, providing a standard support mechanism for authentication schemes such as token cards, Kerberos, Public Key, and S/Key. EAP has been made available in response to increasing demand to augment RAS authentication with third-party security devices.

EAP is fully supported on both the Windows 2000 Dial-Up Server and the Dial-Up Networking Client. EAP is a critical technology component for secure Virtual Private Networks (VPN), protecting them against “brute force” or “dictionary” attacks and password guessing.

EAP improves on previous authentication protocols such as Password Authentication Protocol (PAP) and Challenge Handshake Authentication Protocol (CHAP). Windows 2000 supports these earlier authentication protocols as well.

Tracing

The final chapter in this volume describes the implementation of the common tracing DLL, which provides a uniform mechanism for generating diagnostic output for the Windows NT/Windows 2000 Routing and RAS components (as well as any other application that wishes to use the DLL). The DLL provides dynamic configuration change, allowing a user to direct output to a console or to a specified file.

CHAPTER 3

Using Microsoft Reference Resources

Keeping current with all the latest information on the latest networking technology is like trying to count the packets going through routers at the MAE-WEST Internet service exchange by watching their blinking activity lights: It's impossible. Often times, application developers feel like those routers might feel at a given day's peak activity; too much information is passing through them, none of which is being absorbed or passed along fast enough for their boss' liking.

For developers, sifting through all the *available* information to get to the *required* information is often a major undertaking, and can impose a significant amount of overhead upon a given project. What's needed is either a collection of information that has been sifted for you, shaking out the information you need the most and putting that pertinent information into a format that's useful and efficient, or direction on how to sift the information yourself. The *Networking Services Developer's Reference Library* does the former, and this chapter and the next provide you with the latter.

This veritable white noise of information hasn't always been a problem for network programmers. Not long ago, getting the information you needed was a challenge because there wasn't enough of it; you had to find out where such information might be located and then actually get access to that location, because it wasn't at your fingertips or on some globally available backbone, and such searching took time. In short, the availability of information was limited.

Today, the volume of information that surrounds us sometimes numbs us; we're overloaded with too much information, and if we don't take measures to filter out what we don't need to meet our goals, soon we become inundated and unable to discern what's "white noise" and what's information that we need to stay on top of our respective fields. In short, the overload of available information makes it more difficult for us to find what we *really* need, and wading through the deluge slows us down.

This fact applies equally to Microsoft's reference material, because there is so much information that finding what *you* need can be as challenging as figuring out what to do with it once you have it. Developers need a way to cut through what isn't pertinent to them and to get what they're looking for. One way to ensure you can get to the information you need is to understand the tools you use; carpenters know how to use nail-guns, and it makes them more efficient. Bankers know how to use ten-keys, and it makes them more adept. If you're a developer of Windows applications, two tools you should know are MSDN and MSDN Online. The third tool for developers—reference books from the WPRS—can help you get the most out of the first two.

Books in the WPRS, such as those found in the *Networking Services Developer's Reference Library*, provide reference material that focuses on a given area of Windows programming. MSDN and MSDN Online, in comparison, contain all of the reference material that all Microsoft programming technologies have amassed over the past few years, and create one large repository of information. Regardless of how well such information is organized, there's a lot of it, and if you don't know your way around, finding what you need (even though it's in there, somewhere) can be frustrating, time-consuming, and just an overall bad experience.

This chapter will give you the insight and tips you need to navigate MSDN and MSDN Online and enable you to use each of them to the fullest of their capabilities. Also, other Microsoft reference resources are investigated, and by the end of the chapter, you'll know where to go for the Microsoft reference information you need (and how to quickly and efficiently get there).

The Microsoft Developer Network

MSDN stands for Microsoft Developer Network, and its intent is to provide developers with a network of information to enable the development of Windows applications. Many people have either worked with MSDN or have heard of it, and quite a few have one of the three available subscription levels to MSDN, but there are many, many more who don't have subscriptions and could use some concise direction on what MSDN can do for a developer or development group. If you fall into any of these categories, this section is for you.

There is some clarification to be done with MSDN and its offerings; if you've heard of MSDN, or have had experience with MSDN Online, you may have asked yourself one of these questions during the process of getting up to speed with either resource:

- Why do I need a subscription to MSDN if resources such as MSDN Online are accessible for free over the Internet?
- What is the difference between the three levels of MSDN subscriptions?
- Is there a difference between MSDN and MSDN Online, other than the fact that one is on the Internet and the other is on a CD? Do their features overlap, separate, coincide, or what?

If you have asked any of these questions, then lurking somewhere in the back of your thoughts has probably been a sneaking suspicion that maybe you aren't getting the most out of MSDN. Maybe you're wondering whether you're paying too much for too little, or not enough to get the resources you need. Regardless, you want to be in the know and not in the dark. By the end of this chapter, you'll know the answers to all these questions and more, along with some effective tips and hints on how to make the most effective use of MSDN and MSDN Online.

Comparing MSDN with MSDN Online

Part of the challenge of differentiating between MSDN and MSDN Online comes with determining which has the features you need. Confounding this differentiation is the fact that both have some content in common, yet each offers content unavailable with the other. But can their difference be boiled down? Yes, if broad strokes and some generalities are used:

- MSDN provides reference content *and* the latest Microsoft product software, all shipped to its subscribers on CD or DVD.
- MSDN Online provides reference content *and* a development community forum, and is available only over the Internet.

Each delivery mechanism for the content that Microsoft is making available to Windows developers is appropriate for the medium, and each plays on the strength of the medium to provide its "customers" with the best possible presentation of material. These strengths and medium considerations enable MSDN and MSDN Online to provide developers with different feature sets, each of which has its advantages.

MSDN is perhaps less "immediate" than MSDN Online because it gets to its subscribers in the form of CDs or DVDs that come in the mail. However, MSDN can sit in your CD/DVD drive (or on your hard drive), and isn't subject to Internet speeds or failures. Also, MSDN has a software download feature that enables subscribers to automatically update their local MSDN content over the Internet, as soon as it becomes available, without having to wait for the update CD/DVD to come in the mail. The interface with which MSDN displays its material—which looks a whole lot like a specialized browser window—is also linked to the Internet as a browser-like window. To further coordinate MSDN with the immediacy of the Internet, MSDN Online has a section of the site dedicated to MSDN subscribers that enable subscription material to be updated (on their local machines) as soon as it's available.

MSDN Online has lots of editorial and technical columns that are published directly to the site, and are tailored (not surprisingly) to the issues and challenges faced by developers of Windows applications or Windows-based Web sites. MSDN Online also has a customizable interface (somewhat similar to *MSN.com*) that enables visitors to tailor the information that's presented upon visiting the site to the areas of Windows development in which they are most interested. However, MSDN Online, while full of up-to-date reference material and extensive online developer community content, doesn't come with Microsoft product software, and doesn't reside on your local machine.

Because it's easy to confuse the differences and similarities between MSDN and MSDN Online, it makes sense to figure out a way to quickly identify how and where they depart. Figure 3-1 puts the differences—and similarities—between MSDN and MSDN Online into a quickly identifiable format.

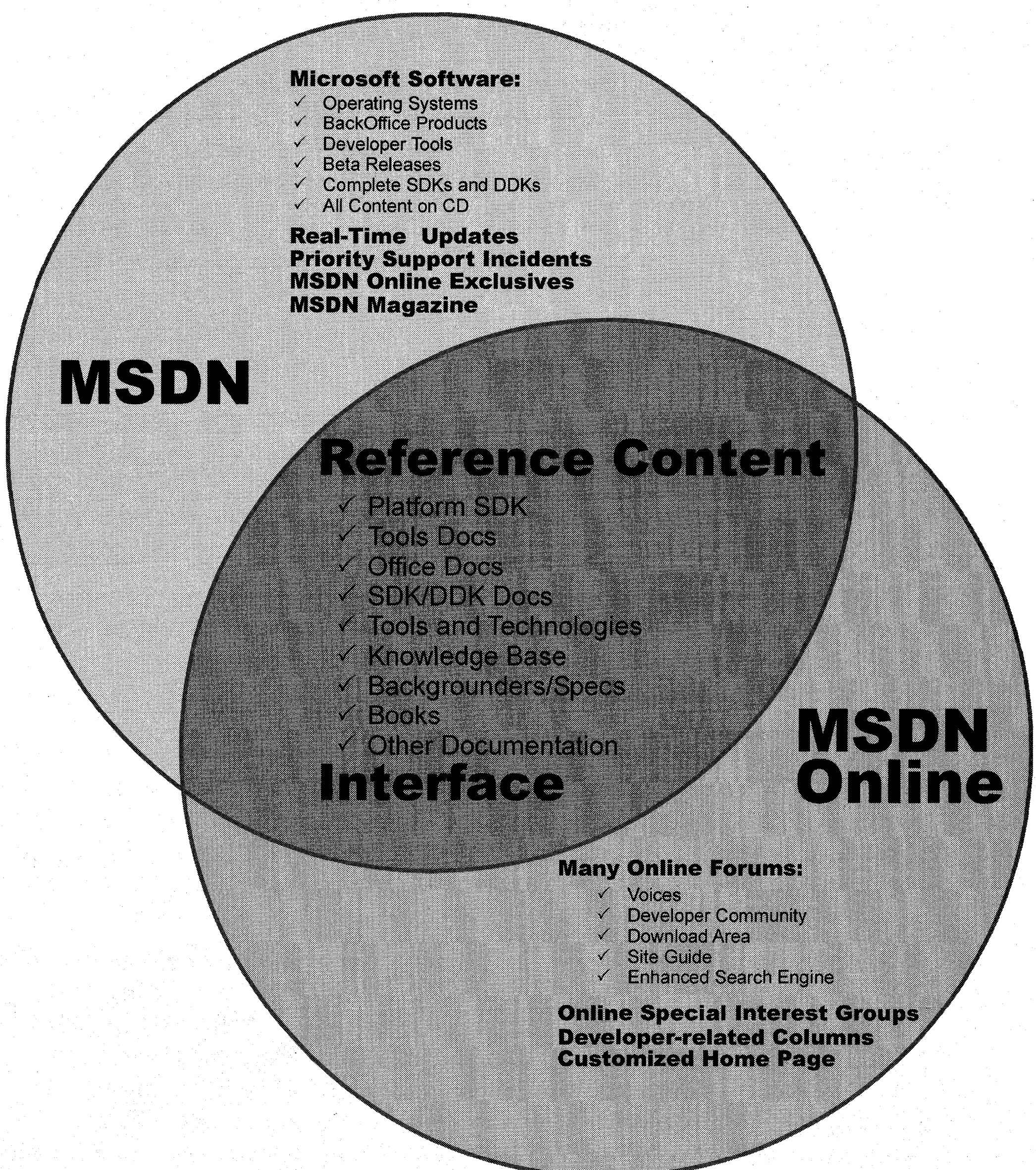


Figure 3-1: The similarities and differences in coverage between MSDN and MSDN Online.

One feature you'll notice is shared between MSDN and MSDN Online is the interface—they are very similar. That's almost certainly a result of attempting to ensure that developers' user experience with MSDN is easily associated with the experience had on MSDN Online, and vice-versa.

Remember, too, that if you are an MSDN subscriber, you can still use MSDN Online and its features. So it isn't an "either/or" question with regard to whether you need an MSDN subscription or whether you should use MSDN Online; if you have an MSDN subscription, you will probably continue to use MSDN Online and the additional features provided with your MSDN subscription.

MSDN Subscriptions

If you're wondering whether you might benefit from a subscription to MSDN, but you aren't quite sure what the differences between its subscription levels are, you aren't alone. This section aims to provide a quick guide to the differences in subscription levels, and even provides an estimate for what each subscription level costs.

The three subscription levels for MSDN are: Library, Professional, and Universal. Each has a different set of features. Each progressive level encompasses the lower level's features, and includes additional features. In other words, with the Professional subscription, you get everything provided in the Library subscription plus additional features; with the Universal subscription, you get everything provided in the Professional subscription plus even more features.

MSDN Library Subscription

The MSDN Library subscription is the basic MSDN subscription. While the Library subscription doesn't come with the Microsoft product software that the Professional and Universal subscriptions provide, it does come with other features that developers may find necessary in their development effort. With the Library subscription, you get the following:

- The Microsoft reference library, including SDK and DDK documentation, updated quarterly
- Lots of sample code, which you can cut-and-paste into your projects, royalty free
- The complete Microsoft Knowledge Base—*the* collection of bugs and workarounds
- Technology specifications for Microsoft technologies
- The complete set of product documentation, such as Microsoft Visual Studio, Microsoft Office, and others
- Complete (and in some cases, partial) electronic copies of selected books and magazines
- Conference and seminar papers—if you weren't there, you can use MSDN's notes

In addition to these items, you also get:

- Archives of MSDN Online columns
- Periodic e-mails from Microsoft chock full of development-related information
- A subscription to MSDN News, a bi-monthly newspaper from the MSDN folks
- Access to subscriber-exclusive areas and material on MSDN Online

MSDN Professional Subscription

The MSDN Professional subscription is a superset of the Library subscription. In addition to the features outlined in the previous section, MSDN Professional subscribers get the following:

- Complete set of Windows operating systems, including release versions of Windows 95, Windows 98, and Windows NT 4 Server and Workstation
- Windows SDKs and DDKs in their entirety
- International versions of Windows operating systems (as chosen)
- Priority technical support for two incidents in a development and test environment

MSDN Universal Subscription

The MSDN Universal subscription is the all-encompassing version of the MSDN subscription. In addition to everything provided in the Professional subscription, Universal subscribers get the following:

- The latest version of Visual Studio, Enterprise Edition
- The Microsoft BackOffice test platform, which includes all sorts of Microsoft product software incorporated in the BackOffice family, each with a special 10-connection license for use in the development of your software products
- Additional development tools, such as Office Developer, Microsoft FrontPage, and Microsoft Project
- Priority technical support for two additional incidents in a development and test environment (for a total of four incidents)

Purchasing an MSDN Subscription

Of course, all the features that you get with MSDN subscriptions aren't free. MSDN subscriptions are one-year subscriptions, which are current as of this writing. Just as each MSDN subscription escalates in functionality of incorporation of features, so does each escalate in price. Please note that prices are subject to change.

The MSDN Library subscription has a retail price of $199, but if you're renewing an existing subscription you get a $100 rebate in the box. There are other perks for existing Microsoft customers, but those vary. Check out the Web site for more details.

The MSDN Professional subscription is a bit more expensive than the Library, with a retail price of $699. If you're an existing customer renewing your subscription, you again get a break in the box, this time in the amount of a $200 rebate. You also get that break if you're an existing Library subscriber who's upgrading to a Professional subscription.

The MSDN Universal subscription takes a big jump in price, sitting at $2,499. If you're upgrading from the Professional subscription, the price drops to $1,999, and if you're upgrading from the Library subscription level, there's an in-the-box rebate for $200.

As is often the case, there are academic and volume discounts available from various resellers, including Microsoft, so those who are in school or in the corporate environment can use their status (as learner or learned) to get a better deal—and in most cases, the deal is in fact much better. Also, if your organization is using lots of Microsoft products, whether or not MSDN is a part of that group, ask your purchasing department to look into the Microsoft Open License program; the Open License program gives purchasing breaks for customers who buy lots of products. Check out *www.microsoft.com/licensing* for more details. Who knows, if your organization qualifies you could end up getting an engraved pen from your purchasing department, or if you're really lucky maybe even a plaque of some sort for saving your company thousands of dollars on Microsoft products.

You can get MSDN subscriptions from a number of sources, including online sites specializing in computer-related information, such as *www.iseminger.com* (shameless self-promotion, I know), or from your favorite online software site. Note that not all software resellers carry MSDN subscriptions; you might have to hunt around to find one. Of course, if you have a local software reseller that you frequent, you can check out whether they carry MSDN subscriptions.

As an added bonus for owners of this *Networking Services Developer's Reference Library*, in the back of Volume 1, you'll find a $200 rebate good toward the purchase of an MSDN Universal subscription. For those of you doing the math, that means you actually *make* money when you purchase the *Networking Services Developer's Reference Library* and an MSDN Universal subscription. With this rebate, every developer in your organization can have the *Networking Services Developer's Refence Library* on their desk and the MSDN Universal subscription on thier desktop, and still come out $50 ahead. That's the kind of math even accountants can like.

Using MSDN

MSDN subscriptions come with an installable interface, and the Professional and Universal subscriptions also come with a bunch of Microsoft product software such as Windows platform versions and BackOffice applications. There's no need to tell you how to use Microsoft product software, but there's a lot to be said for providing some quick but useful guidance on getting the most out of the interface to present and navigate through the seemingly endless supply of reference material provided with any MSDN subscription.

To those who have used MSDN, the interface shown in Figure 3-2 is likely familiar; it's the navigational front-end to MSDN reference material.

The interface is familiar and straightforward enough, but if you don't have a grasp on its features and navigation tools, you can be left a little lost in its sea of information. With a few sentences of explanation and some tips for effective navigation, however, you can increase its effectiveness dramatically.

Navigating MSDN

One of the primary features of MSDN—and to many, its primary drawback—is the sheer volume of information it contains, over 1.1GB and growing. The creators of MSDN likely realized this, though, and have taken steps to assuage the problem. Most of those steps relate to enabling developers to selectively navigate through MSDN's content.

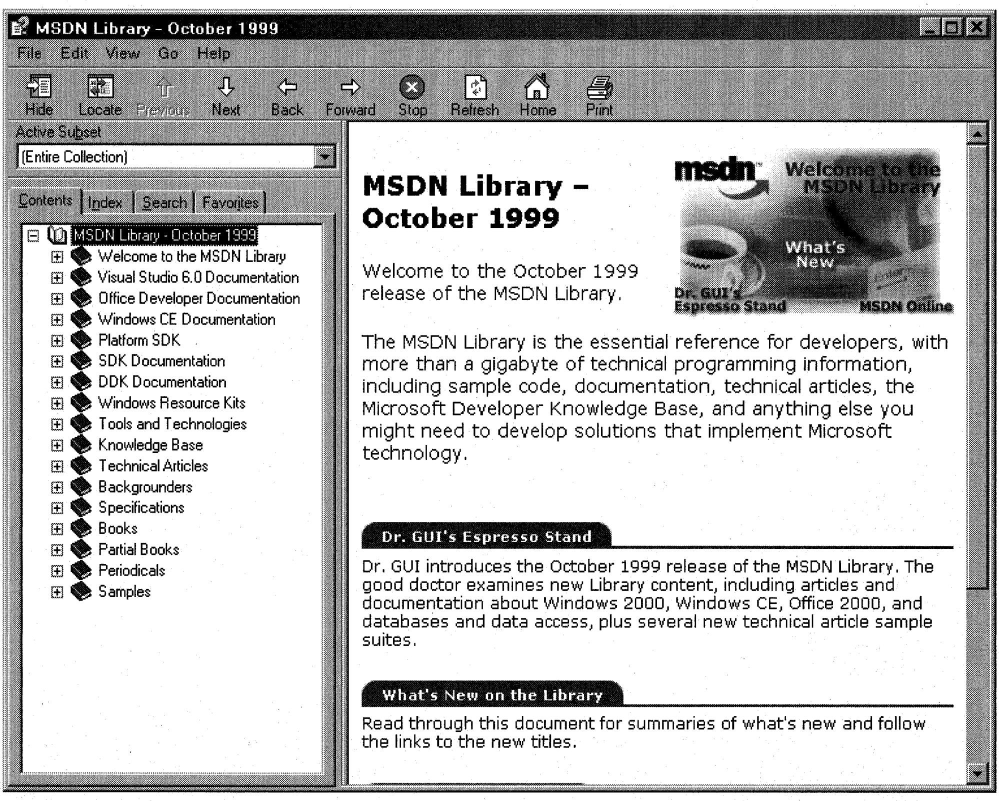


Figure 3-2: The MSDN interface.

Basic navigation through MSDN is simple and is a lot like navigating through Microsoft Windows Explorer and its folder structure. Instead of folders, MSDN has books into which it organizes its topics; expand a book by clicking the + box to its left, and its contents are displayed with its nested books or reference pages, as shown in Figure 3-3. If you don't see the left pane in your MSDN viewer, go to the View menu and select Navigation Tabs and they'll appear.

The four tabs in the left pane of MSDN—increasingly referred to as property sheets these days—are the primary means of navigating through MSDN content. These four tabs, in coordination with the Active Subset drop-down box above the four tabs, are the tools you use to search through MSDN content. When used to their full extent, these coordinated navigation tools greatly improve your MSDN experience.

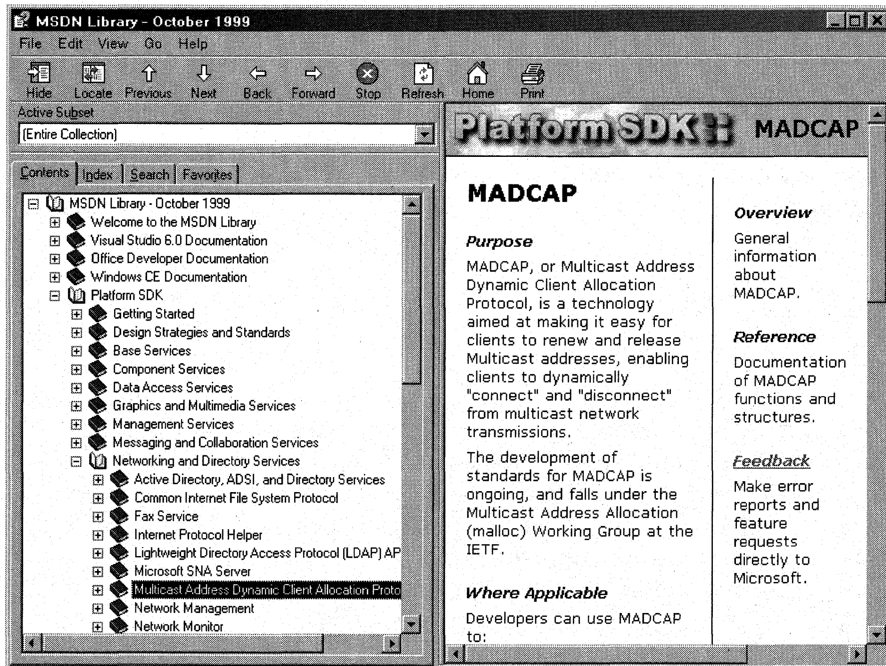


Figure 3-3: Basic navigation through MSDN.

The Active Subset drop-down box is a filter mechanism; choose the subset of MSDN information you're interested in working with from the drop-down box, and the information in each of the four Navigation Tabs (including the Contents tab) limits the information it displays to the information contained in the selected subset. This means that any searches you do in the Search tab, and in the index presented in the Index tab, are filtered by their results and/or matches to the subset you define, greatly narrowing the number of potential results for a given inquiry. This enables you to better find the information you're *really* looking for. In the Index tab, results that might match your inquiry but *aren't* in the subset you have chosen are grayed out (but still selectable). In the Search tab, they simply aren't displayed.

MSDN comes with the following predefined subsets (these subsets are subject to change, based on documentation updates and TOC reorganizations):

Entire Collection
MSDN, Books and Periodicals
MSDN, Content on Disk 2 only (CD only – not in DVD version)
MSDN, Content on Disk 3 only (CD only – not in DVD version)
MSDN, Knowledge Base
MSDN, Technical Articles and Backgrounders
Platform SDK, Networking Services
Platform SDK, Security
Platform SDK, Tools and Languages
Platform SDK, User Interface Services
Platform SDK, Web Services
Platform SDK, Win32 API
Repository 2.0 Documentation
Visual Basic Documentation
Visual C++ Documentation

Office Developer Documentation
Platform SDK, BackOffice
Platform SDK, Base Services
Platform SDK, Component Services
Platform SDK, Data Access Services
Platform SDK, Getting Started
Platform SDK, Graphics and Multimedia Services
Platform SDK, Management Services
Platform SDK, Messaging and Collaboration Services
Visual C++, Platform SDK and WinCE Docs
Visual C++, Platform SDK, and Enterprise Docs
Visual FoxPro Documentation
Visual InterDev Documentation
Visual J++ Documentation
Visual SourceSafe Documentation
Visual Studio Product Documentation
Windows CE Documentation

As you can see, these filtering options essentially mirror the structure of information delivery used by MSDN. But what if you are interested in viewing the information in a handful of these subsets? For example, what if you want to search on a certain keyword through the Platform SDK's ADSI, Networking Services, and Management Services subsets, as well as a little section that's nested way into the Base Services subset? Simple—you define your own subset by choosing the View menu, and then selecting the Define Subsets menu item. You're presented with the window shown in Figure 3-4.

Defining a subset is easy; just take the following steps:

1. Choose the information you want in the new subset; you can choose entire subsets or selected books/content within available subsets.
2. Add your selected information to the subset you're creating by clicking the Add button.
3. Name the newly created subset by typing in a name in the Save New Subset As box. Note that defined subsets (including any you create) are arranged in alphabetical order.

You can also delete entire subsets from the MSDN installation. Simply select the subset you want to delete from the Select Subset To Display drop-down box, and then click the nearby Delete button.

Once you have defined a subset, it becomes available in MSDN just like the predefined subsets, and filters the information available in the four Navigation Tabs, just like the predefined subsets do.

Quick Tips

Now that you know how to navigate MSDN, there are a handful of tips and tricks that you can use to make MSDN as effective as it can be.

Use the Locate button to get your bearings. Perhaps it's human nature to need to know where you are in the grand scheme of things, but regardless, it can be bothersome to have a reference page displayed in the right pane (perhaps jumped to from a search), without the Contents tab in the left pane being synchronized in terms of the reference page's location in the information tree. Even if you know the general technology in which your reference page resides, it's nice to find out where it is in the content structure.

This is easy to fix. Simply click the Locate button in the navigation toolbar and all will be synchronized.

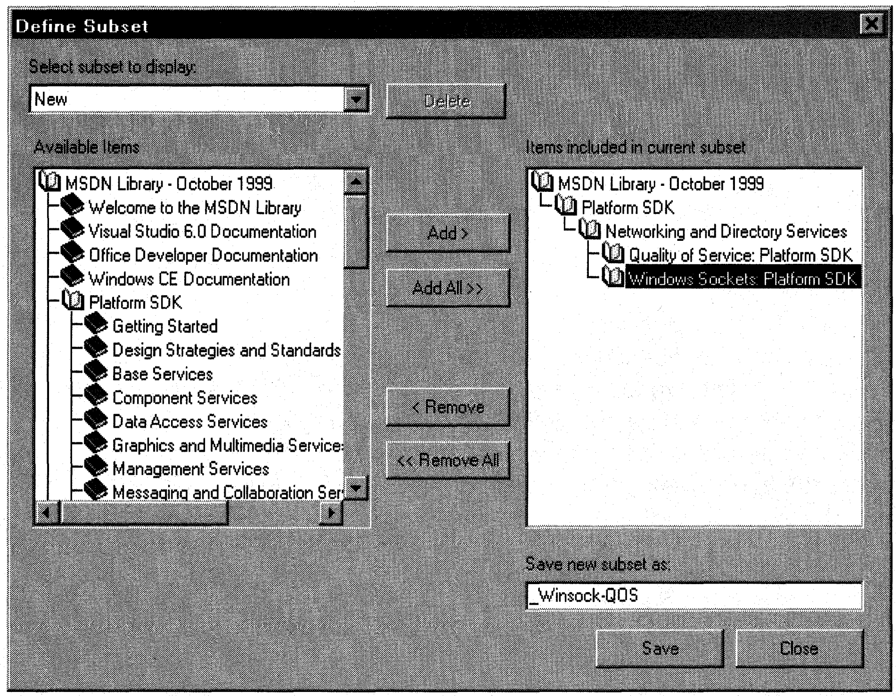


Figure 3-4: The Define Subsets window.

Use the Back button just like a browser. The Back button in the navigation toolbar functions just like a browser's Back button; if you need information on a reference page you viewed previously, you can use the Back button to get there, rather than going through the process of doing another search.

Define your own subsets, and use them. Like I said at the beginning of this chapter, the volume of information available these days can sometimes make it difficult to get our work done. By defining subsets of MSDN that are tailored to the work you do, you can become more efficient.

Use an underscore at the beginning of your named subsets. Subsets in the Active Subset drop-down box are arranged in alphabetical order, and the drop-down box shows only a few subsets at a time (making it difficult to get a grip on available subsets, I think). Underscores come before letters in alphabetical order, so if you use an underscore on all of your defined subsets, you get them placed at the front of the Active Subset listing of available subsets. Also, by using an underscore, you can immediately see which subsets you've defined, and which ones come with MSDN—it saves a few seconds at most, but those seconds can add up.

Using MSDN Online

MSDN underwent a redesign in December of 1999, aimed at streamlining the information provided, jazzing things up with more color, highlighting hot new technologies, and various other improvements. Despite its visual overhaul, MSDN Online still shares a lot of content and information delivery similarities with MSDN, and those similarities are by design; when you can go from one developer resource to another and immediately work with its content, your job is made easier. However, MSDN Online is different enough that it merits explaining in its own right—it's a different delivery medium, and can take advantage of the Internet in ways that MSDN simply cannot.

If you've used MSN's home page before (*www.msn.com*), you're familiar with the fact that you can customize the page to your liking; choose from an assortment of available national news, computer news, local news, local weather, stock quotes, and other collections of information or news that suit your tastes or interests. You can even insert a few Web links and have them readily accessible when you visit the site. The MSDN Online home page can be customized in a similar way, but its collection of headlines, information, and news sources are all about development. The information you choose specifies the information you see when you go to the MSDN Online home page, just like the MSN home page.

There are a couple of ways to get to the customization page; you can go to the MSDN Online home page (*msdn.microsoft.com*) and click the Personalize This Site button near the top of the page, or you can go there directly by pointing your browser to *msdn.microsoft.com/msdn-online/start/custom*. However you get there, the page you'll see is shown in Figure 3-5.

As you can see from Figure 3-5, there are lots of technologies to choose from (many more options can be found when you scroll down through available technologies). If you're interested in Web development, you can select the checkbox at the left of the page next to Standard Web Development, and a predefined subset of Web-centered technologies is selected. For technologies centered more on Network Services, you can go through and choose the appropriate technologies. If you want to choose all the technologies in a given technology group more quickly, click the Select All button in the technology's shaded title area.

You can also choose which tab is selected by default in the home page that MSDN Online presents to you, which is convenient for dropping you into the category of MSDN Online information that interests you most. All five of the tabs available on MSDN Online's home page are available for selection; those tabs are the following:

- Features
- News
- Columns
- Technical Articles
- Training & Events

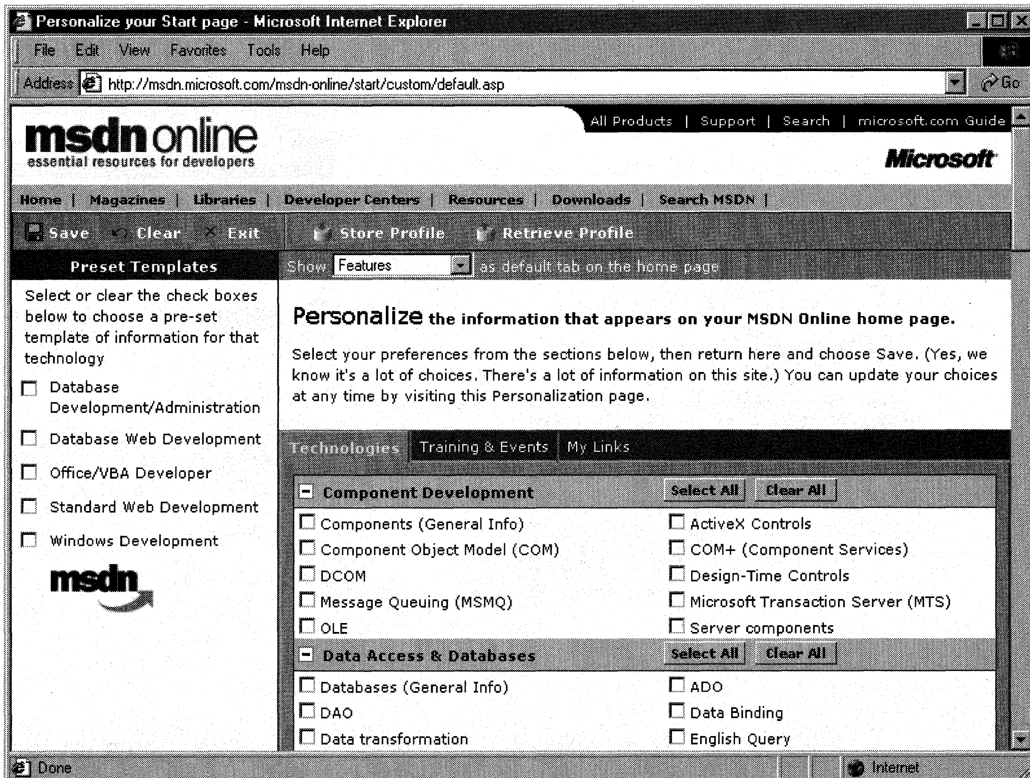


Figure 3-5: The MSDN Online Personalize Page.

Once you've defined your profile—that is, customized the MSDN Online content you want to see—MSDN Online shows you the most recent information pertinent to your profile when you go to MSDN Online's home page, with the default tab you've chosen displayed upon loading of the MSDN Online home page.

Finally, if you want your profile to be available to you regardless of which computer you're using, you can direct MSDN Online to store your profile. Storing a profile for MSDN Online results in your profile being stored on MSDN Online's server, much like roaming profiles in Windows 2000, and thereby makes your profile available to you regardless of the computer you're using. The option of storing your profile is available when you customize your MSDN Online home page (and can be done any time thereafter). The storing of a profile, however, requires that you become a registered member of MSDN Online. More information about becoming a registered MSDN Online user is provided in the section titled *MSDN Online Registered Users*.

Navigating MSDN Online

Once you're done customizing the MSDN Online home page to get the information you're most interested in, navigating through MSDN Online is easy. A banner that sits just below the MSDN Online logo functions as a navigation bar, with drop-down menus that can take you to the available areas on MSDN Online, as Figure 3-6 illustrates.

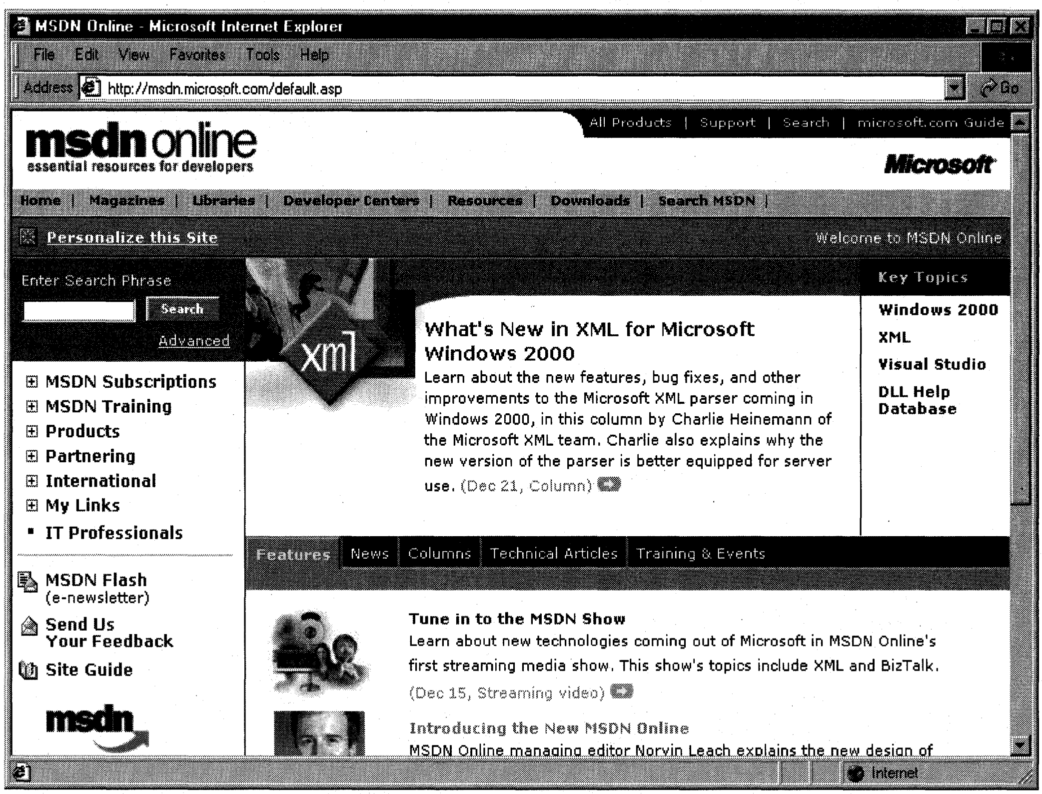


Figure 3-6: The MSDN Online Navigation Bar with Its Drop-Down Menus.

Following is a list of available menu categories, which groups the available sites and features within MSDN Online:

Home	Resources
Magazines	Downloads
Libraries	Search MSDN
Developer Centers	

The navigation bar is available regardless of where you are in MSDN Online, so the capability to navigate the site from this familiar menu is always available, leaving you a click away from any area on MSDN Online. These menu categories create a functional and logical grouping of MSDN Online's feature offerings.

MSDN Online Features

Each of MSDN Online's seven feature categories contains various sites that comprise the features available to developers visiting MSDN Online.

Home is already familiar; clicking on Home in the navigation bar takes you to the MSDN Online home page that you've (perhaps) customized, showing you all the latest information about technologies that you've indicated you're interested in reading about.

Magazines is a collection of columns and articles that comprise MSDN Online's magazine section, as well as online versions of Microsoft's magazines such as MSJ, MIND, and the MSDN Show (a Webcast feature introduced with the December 1999 remodeling of MSDN Online). The Magazines feature of MSDN Online can be linked to directly at *msdn.microsoft.com/resources/magazines.asp*. The Magazines home page is shown in Figure 3-7.

Figure 3-7: The Magazines Home Page.

For those of you familiar with the **Voices** feature section that formerly found its home on the MSDN Online navigation banner, don't worry; all content formerly in the Voices section is included the Magazines section as a subsite (or menu item, if you prefer) of the Magazines site. For those of you who aren't familiar with the Voices subsite, you'll find a bunch of different articles or "voices" there, each of which adds its own particular twist on the issues that face developers. Both application and Web developers can get their fill of magazine-like articles from the sizable list of different articles available (and frequently refreshed) in the Voices subsite. With the combination of columns and online developer magazines offered in the Magazines section, you're sure to find plenty of interesting insights.

Libraries is where the reference material available on MSDN Online lives. The Libraries site is divided into two sections: Library and Web Workshop. This distinction divides the reference material between Windows application development and Web development. Choosing Library from the Libraries menu takes you to a page through which you can navigate in traditional MSDN fashion, and gain access to traditional MSDN reference material. The Library home page can be linked to directly at *msdn.microsoft.com/library*. Choosing Web Workshop takes you to a site that enables you to navigate the Web Workshop in a slightly different way, starting with a bulleted list of start points, as shown in Figure 3-8. The Web Workshop home page can be linked to directly at *msdn.microsoft.com/workshop*.

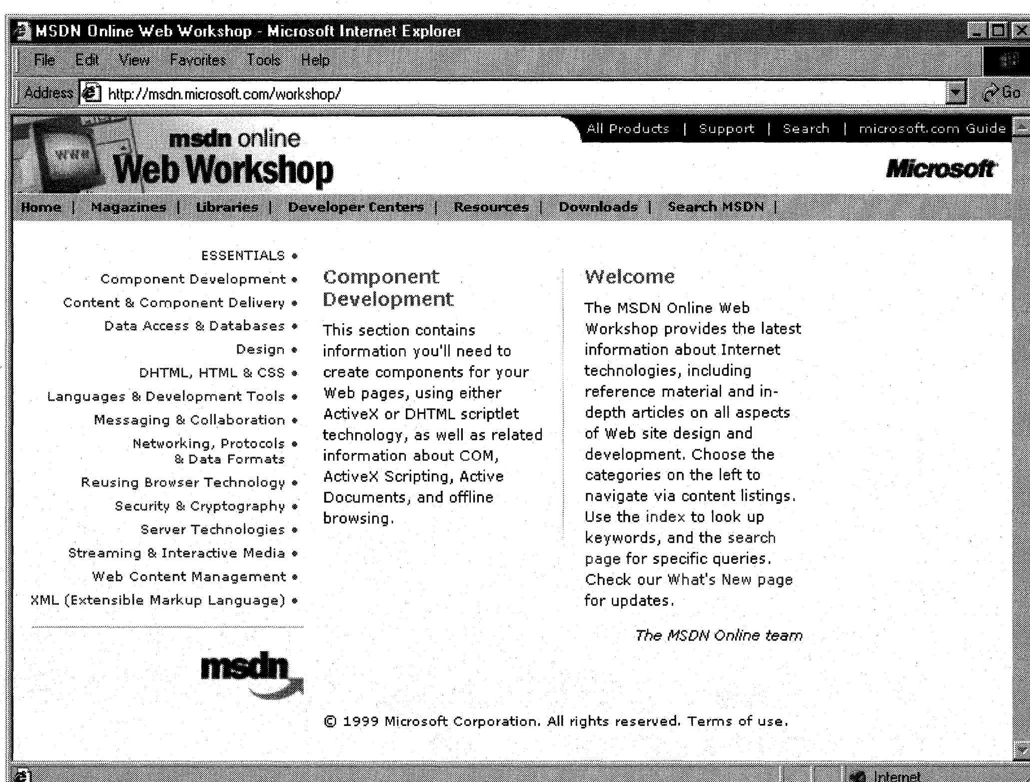


Figure 3-8: The Web Workshop Home Page.

Developer Centers is a hub from which developers who are interested in a particular area of development—such as Windows 2000, SQL Server, or XML—can go to find focused Web site centers within MSDN Online. Each developer center is dedicated to providing all sorts of information associated with its area of focus. For example, the Windows 2000 developer center has information about what's new with Windows 2000, including newsgroups, specifications, chats, knowledge base articles, and news, among others. At publication time, MSDN Online had the following developer centers:

- Microsoft Windows 2000
- Microsoft Exchange
- Microsoft SQL Server
- Microsoft Windows Media
- XML

In addition to these developer centers is a promise that new centers would be added to the site in the future. To get to the Developer Centers home page directly, link to *msdn.microsoft.com/resources/devcenters.asp*. Figure 3-9 shows the Developer Centers home page.

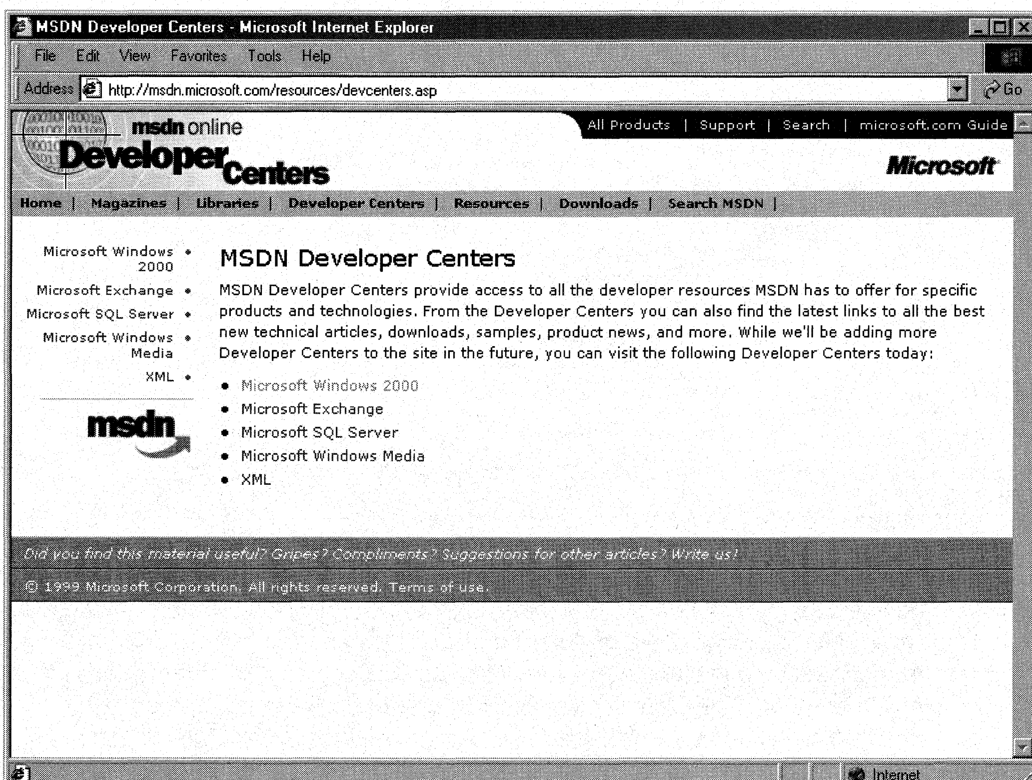


Figure 3-9: The Developer Centers Home Page.

Resources is a place where developers can go to take advantage of the online forum of Windows and Web developers, in which ideas or techniques can be shared, advice can be found or given (through MHM, or Members Helping Members), and the MSDN User Group Program can be joined or perused to find a forum to voice their opinions or chat with other developers. The Resources site is full of all sorts of useful stuff, including featured books, a DLL help database, online chats, case studies, and more. The Resources home page can be linked to directly at *msdn.microsoft.com/resources*. Figure 3-10 provides a look at the Resources home page.

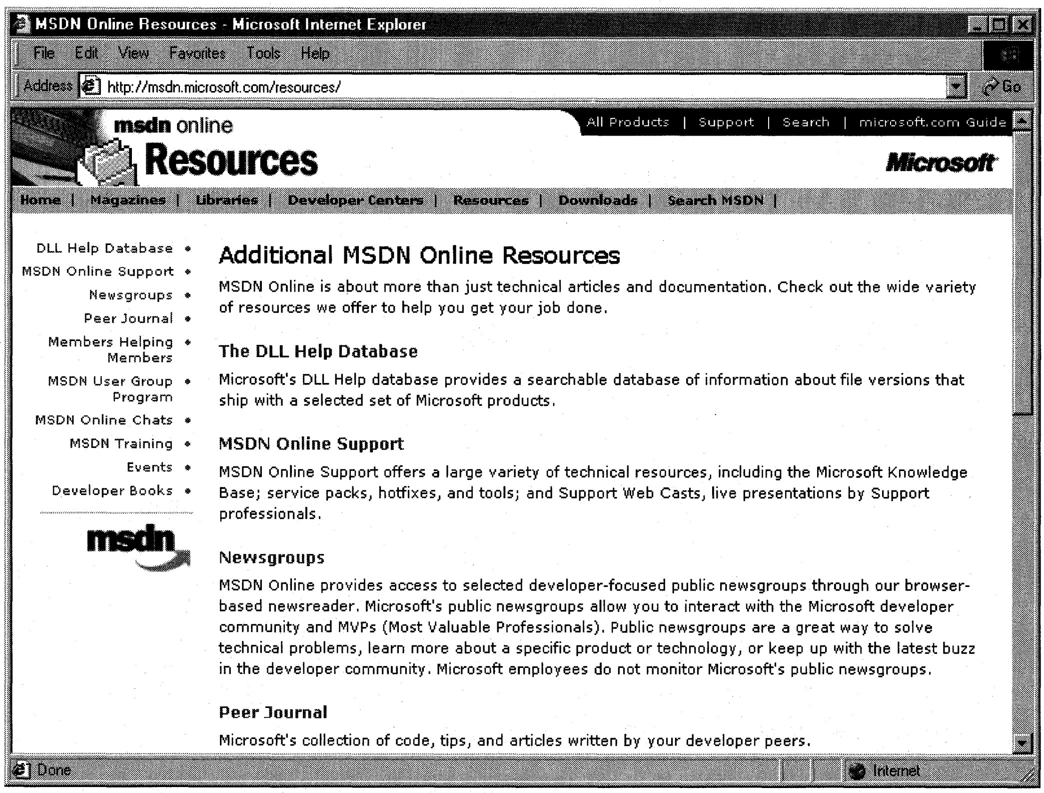


Figure 3-10: The Resources Home Page.

The **Downloads** site is where developers can find all sorts of useable items fit to be downloaded, such as tools, samples, images, and sounds. The Downloads site is also where MSDN subscribers go to get their subscription content updated over the Internet to the latest and greatest releases, as described previously in this chapter in the *Using MSDN* section. The Downloads home page can be linked to directly at *msdn.microsoft.com/downloads*. The Downloads home page is shown in Figure 3-11.

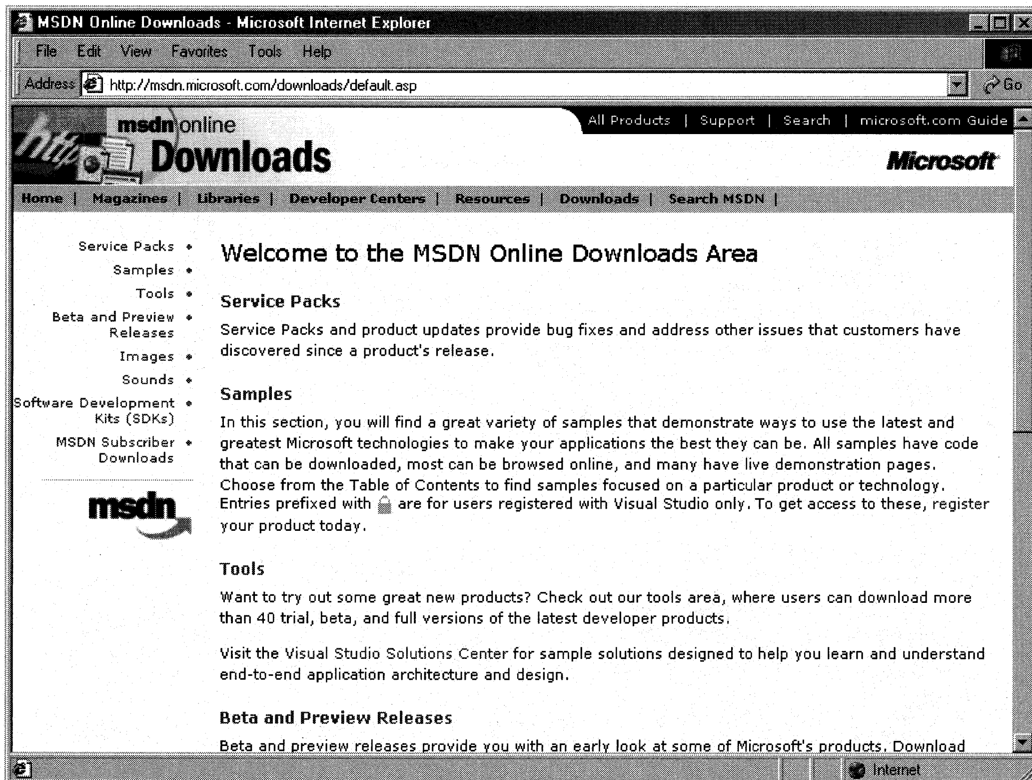


Figure 3-11: The Downloads Home Page.

The **Search MSDN** site on MSDN Online has been improved over previous versions, and includes the capability to restrict searches to either library (Library or Web Workshop), as well as other fine-tune search capabilities. The Search MSDN home page can be linked to directly at *msdn.microsoft.com/search*. The Search MSDN home page is shown in Figure 3-12.

There are two other destinations within MSDN Online of specific interest, neither of which is immediately reachable through the MSDN navigation bar. The first is the **MSDN Online Member Community** home page, and the other is the **Site Guide**.

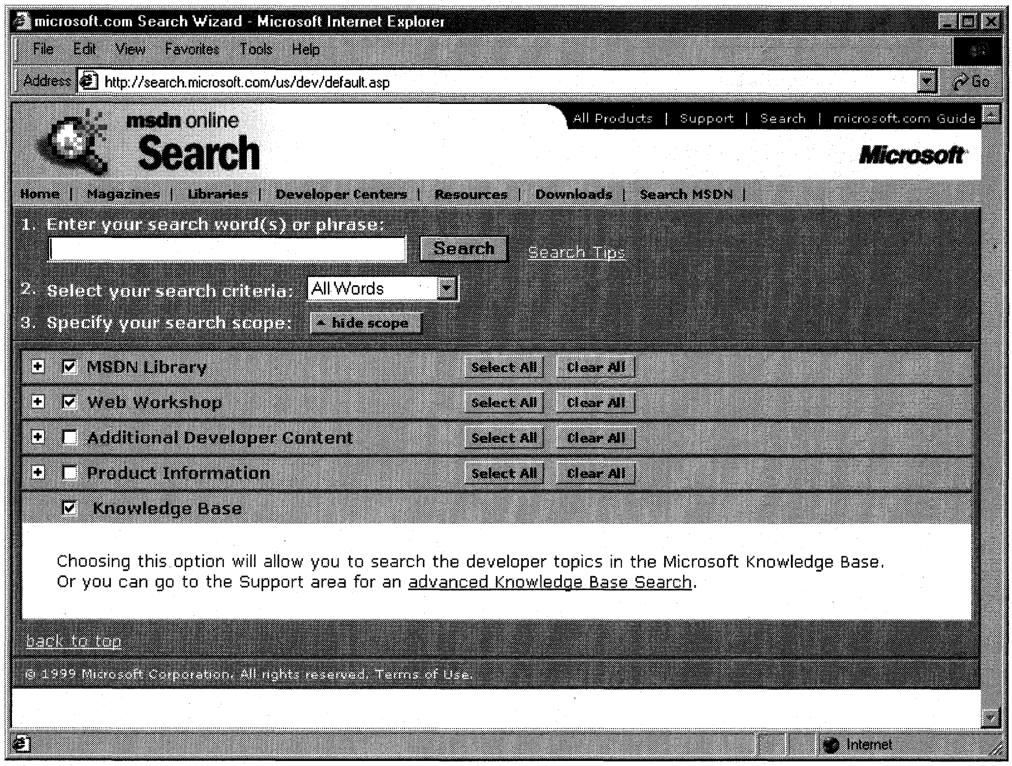


Figure 3-12: The Search MSDN Home Page.

The MSDN Online Member Community home page can be directly reached at *msdn.microsoft.com/community*. Many of the features found in the **Resources** navigation menu are actually subsites of the Community page. Of course, becoming a member of the MSDN Online member community requires that you register (see the next section for more details on joining), but doing so enables you to get access to Online Special Interest Groups (OSIGs) and other features reserved for registered members. The Community page is shown in Figure 3-13.

Another destination of interest on MSDN Online that isn't displayed on the navigation banner is the **Site Guide**. The Site Guide is just what its name suggests—a guide to the MSDN Online site that aims at helping developers find items of interest, and includes links to other pages on MSDN Online such as a recently posted files listing, site maps, glossaries, and other useful links. The Site Guide home page can be linked to directly at *msdn.microsoft.com/siteguide*.

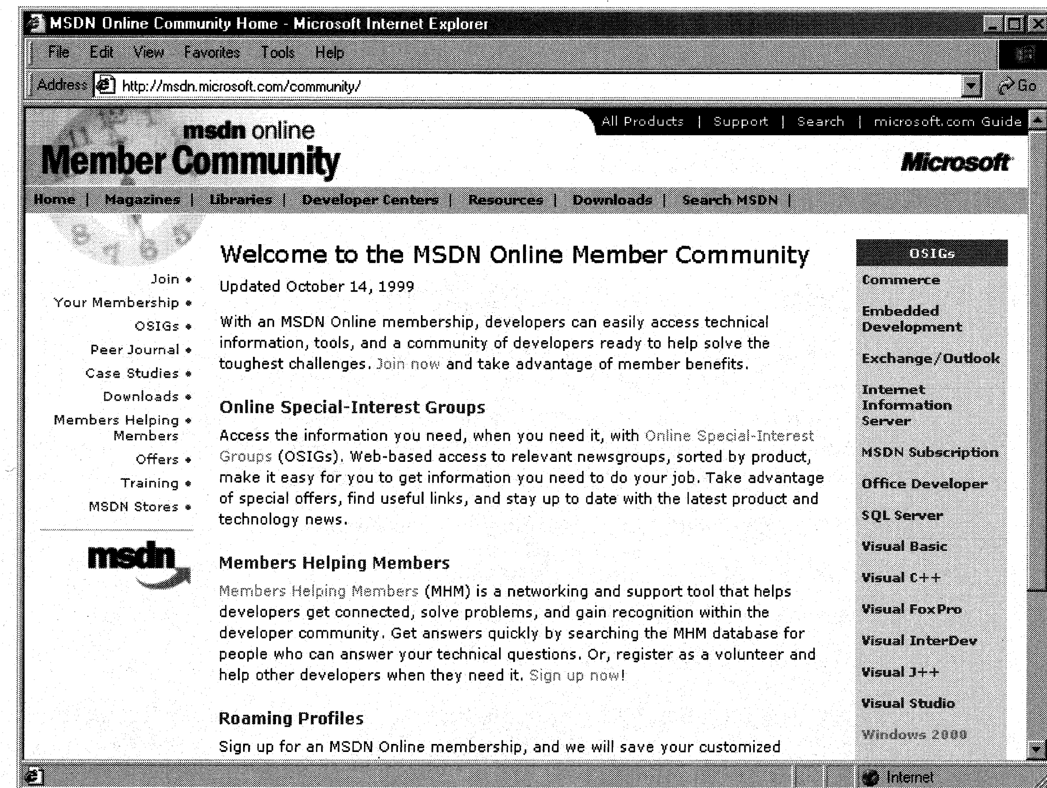


Figure 3-13: The MSDN Online Member Community Home Page.

MSDN Online Registered Users

You may have noticed that some features of MSDN Online—such as the capability to create a store profile of the entry ticket to some community features—require you to become a registered user. Unlike MSDN subscriptions, becoming a registered user of MSDN Online won't cost you anything more but a few minutes of registration time.

Some features of MSDN Online require registration before you can take advantage of their offerings. For example, becoming a member of an OSIG requires registration. That feature alone is enough to register; rather than attempting to call your developer buddy for an answer to a question (only to find out that she's on vacation for two days, and your deadline is in a few hours), you can go to MSDN Online's Community site and ferret through your OSIG to find the answer in a handful of clicks. Who knows; maybe your developer buddy will begin calling you with questions—you don't have to tell her where you're getting all your answers.

There are a number of advantages to being a registered user, such as the choice to receive newsletters right in your inbox if you want to. You can also get all sorts of other timely information, such as chat reminders that let you know when experts on a given subject will be chatting in the MSDN Online Community site. You can also sign up to get newsletters based on your membership in various OSIGs—again, only if you want to. It's easy for me to suggest that you become a registered user for MSDN Online—I'm a registered user, and it's a great resource.

The Windows Programming Reference Series

The WPRS provides developers with timely, concise, and focused material on a given topic, enabling developers to get their work done as efficiently as possible. In addition to providing reference material for Microsoft technologies, each Library in the WPRS also includes material that helps developers get the most out of its technologies, and provides insights that might otherwise be difficult to find.

The WPRS currently includes the following libraries:

- *Microsoft Win32 Developer's Reference Library*
- *Active Directory Developer's Reference Library*
- *Networking Services Developer's Reference Library*

In the near future (subject, of course, to technology release schedules, demand, and other forces that can impact publication decisions), you can look for these prospective WPRS Libraries that cover the following material:

- Web Technologies Library
- Web Reference Library
- MFC Developer's Reference Library
- Com Developer's Reference Library

What else might you find in the future? Planned topics such as a Security Library, Programming Languages Reference Library, BackOffice Developer's Reference Library, or other pertinent topics that developers using Microsoft products need in order to get the most out of their development efforts, are prime subjects for future membership in the WPRS. If you have feedback you want to provide on such libraries, or on the WPRS in general, you can send email to *winprs@microsoft.com*.

If you're sending mail about a particular library, make sure you put the name of the library in the subject line. For example, e-mail about the *Networking Services Developer's Reference Library* would have a subject line that reads "*Networking Services Developer's Reference Library*." There aren't any guarantees that you'll get a reply, but I'll read all of the mail and do what I can to ensure your comments, concerns, or (especially) compliments get to the right place.

CHAPTER 4

Finding the Developer Resources You Need

Networking is complex, and its resource information vast. With all the resources available for developers of network-enabled applications, and the answers they can provide to questions or problems that developers face every day, finding the developer information you need can be a challenge. To address that problem, this chapter is designed to be your one-stop resource to find the developer resources you need, making the job of actually developing your application just a little easier.

Microsoft provides plenty of resource material through MSDN and MSDN Online, and the WPRS provides a great filtered version of focused reference material and development knowledge. However, there is a lot more information to be had. Some of that information comes from Microsoft, some of it from the general development community, and yet more information comes from companies that specialize in such development services. Regardless of which resource you choose, in this chapter you can find out what your development resource options are, and be more informed about the resources that are available to you.

Microsoft provides developer resources through a number of different media, channels, and approaches. The extensiveness of Microsoft's resource offerings mirrors the fact that many are appropriate under various circumstances. For example, you wouldn't go to a conference to find the answer to a specific development problem in your programming project; instead, you might use one of the other Microsoft resources.

Developer Support

Microsoft's support sites cover a wide variety of support issues and approaches, including all of Microsoft's products, but most of those sites are not pertinent to developers. Some sites, however, *are* designed for developer support; the Product Services Support page for developers is a good central place to find the support information you need. Figure 4-1 shows the Product Services Support page for developers, which can be reached at *www.microsoft.com/support/customer/develop.htm*.

Note that there are a number of options for support from Microsoft, including everything from simple online searches of known bugs in the Knowledge Base to hands-on consulting support from Microsoft Consulting Services, and everything in between. The Web page displayed in Figure 4-1 is a good starting point from which you can find out more information about Microsoft's support services.

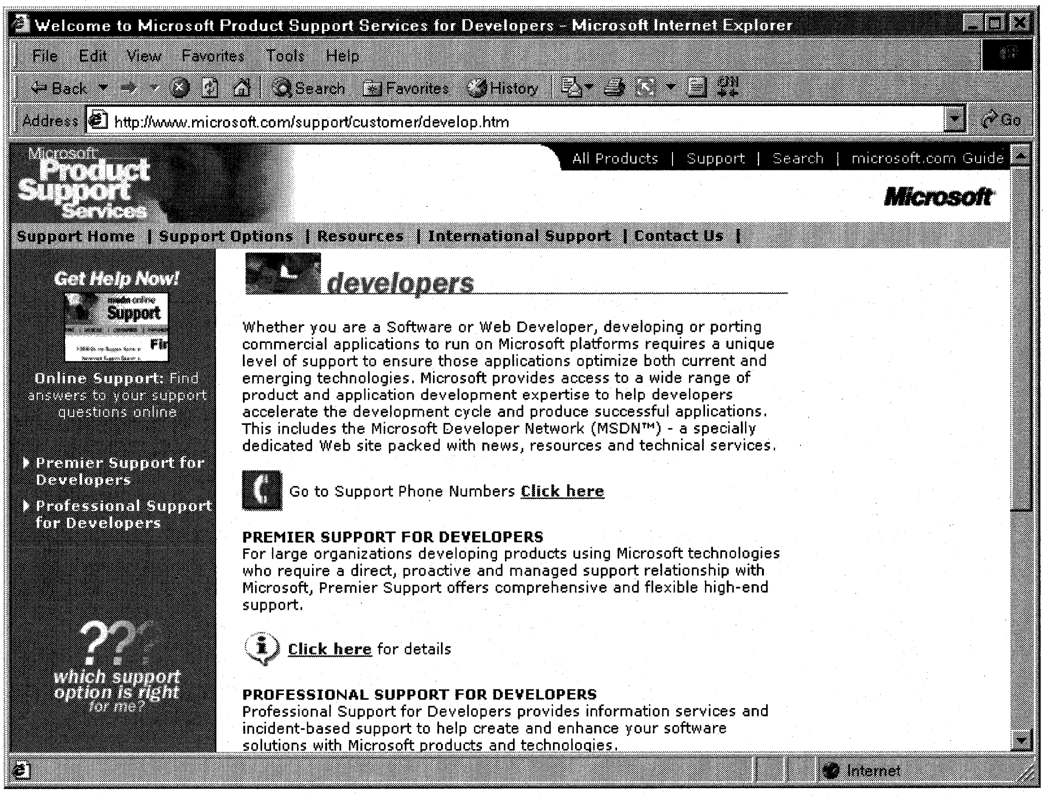


Figure 4-1: The Product Services Support page for developers.

Premier Support from Microsoft provides extensive support for developers, and includes different packages geared toward specific Microsoft customer needs. The packages of Premier Support that Microsoft provides are:

- Premier Support for Enterprises
- Premier Support for Developers
- Premier Support for Microsoft Certified Solution Providers
- Premier Support for OEMs

If you're a developer, you could fall into any of these categories. To find out more information about Microsoft's Premier Support, contact them at (800) 936-2000.

Priority Annual Support from Microsoft is geared toward developers or organizations that have more than an occasional need to call Microsoft with support questions and need priority handling of their support questions or issues. There are three packages of Priority Annual Support offered by Microsoft.

- Priority Comprehensive Support
- Priority Developer Support
- Priority Desktop Support

The best support option for you as a developer is the Priority Developer support. To obtain more information about Priority Developer Support, call Microsoft at (800) 936-3500.

Microsoft also offers a **Pay-Per-Incident Support** option so you can get help if there's just one question that you must have answered. With Pay-Per-Incident Support, you call a toll-free number and provide your Visa, MasterCard, or American Express account number, after which you receive support for your incident. In loose terms, an incident is a problem or issue that can't be broken down into subissues or subproblems (that is, it can't be broken down into smaller pieces). The number to call for Pay-Per-Incident Support is (800) 936-5800.

Note that Microsoft provides two priority technical support incidents as part of the MSDN Professional subscription, and provides four priority technical support incidents as part of the MSDN Universal subscription.

You can also **submit questions** to Microsoft engineers through Microsoft's support Web site, but if you're on a time line you might want to rethink this approach and consider going to MSDN Online and looking into the Community site for help with your development question. To submit a question to Microsoft engineers online, go to *support.microsoft.com/support/webresponse.asp*.

Online Resources

Microsoft also provides extensive developer support through its community of developers found on MSDN Online. At MSDN Online's Community site, you will find OSIGs that cover all sorts of issues in an online, ongoing fashion. To get to MSDN Online's Community site, simply go to *msdn.microsoft.com/community*.

Microsoft's MSDN Online also provides its **Knowledge Base** online, which is part of the Personal Support Center on Microsoft's corporate site. You can search the Knowledge Base online at *support.microsoft.com/support/search*.

Microsoft provides a number of **newsgroups** that developers can use to view information on newsgroup-specific topics, providing yet another developer resource for information about creating Windows applications. To find out which newsgroups are available and how to get to them, go to *support.microsoft.com/support/news*.

The following newsgroups will probably be of particular interest to readers of the *Active Directory Developer's Reference Library*:

- *microsoft.public.win2000.**
- *microsoft.public.msdn.general*
- *microsoft.public.platformsdk.active.directory*
- *microsoft.public.platformsdk.adsi*

- *microsoft.public.platformsdk.dist_svcs*
- *microsoft.public.vb.**
- *microsoft.public.vc.**
- *microsoft.public.vstudio.*microsoft.public.cert.**
- *microsoft.public.certification.**

Of course, Microsoft isn't the only newsgroup provider on which newsgroups pertaining to developing on Windows are hosted. Usenet has all sorts of newsgroups—too many to list—that host ongoing discussions pertaining to developing applications on the Windows platform. You can access newsgroups on Windows development just as you access any other newsgroup; generally, you'll need to contact your ISP to find out the name of the mail server and then use a newsreader application to visit, read, or post to the Usenet groups.

For network developers with a taste for Winsock (and QOS) programming, another site of interest is *www.stardust.com*, which is chock full of up-to-date information about Winsock development and other network-related information. There's other information about network programming on the site, so it's worth a look.

Internet Standards

Many of the network protocols and services implemented in Windows platforms conform to one or more Internet standards recommendations that have gone through a process of review and comments. One especially useful source of information about such standards, recommendations, and ongoing comment periods is the Internet Engineering Task Force, or IETF. Rather than go into some long-winded (page-eating) explanation of what the IETF is, does, and stands for, let me simply say that this is the place where networking protocols and other various Internet-related services are often born, scrutinized, recast, commented upon, and although not standardized or implemented, recommended in a final form called a request for comment, or RFC, even though it's essentially a standard by the time it gets to RFC stage.

If you want to get a clear technical picture of a given technology or protocol, or if you're inclined to comment on the creation and subsequent scrutiny of such things, the place you should go is *www.ietf.org*. This site can tell you all you want to know about the goings on of the IETF, their (non-profit) mission, their Working Groups, and all the information you might ever want about almost anything that has to do with networking recommendations.

If you're curious about a given protocol or networking technology, and want to find an unadulterated (albeit technical) version of its explanation, this is a great place to go. It's a virtual hangout for the brightest people in networking, and it's worth a look or two, even just for the sake of satisfying curiosity.

Learning Products

Microsoft provides a number of products that enable developers to get versed in the particular tasks or tools that they need to achieve their goals (or to finish their tasks). One product line that is geared toward developers is called the Mastering series, and its products provide comprehensive, well-structured interactive teaching tools for a wide variety of development topics.

The Mastering Series from Microsoft contains interactive tools that group books and CDs together so that you can master the topic in question, and there are products available based on the type of application you're developing. To obtain more information about the Mastering series of products, or to find out what kind of offerings the Mastering series has, check out *msdn.microsoft.com/mastering*.

Other learning products are available from other vendors as well, such as other publishers, other application providers that create tutorial-type content and applications, and companies that issue videos (both taped and broadcast over the Internet) on specific technologies. For one example of a company that issues technology-based instructional or overview videos, take a look at *www.compchannel.com*.

Another way of learning about development in a particular language (such as C++, FoxPro, or Microsoft Visual Basic), for a particular operating system, or for a particular product (such as Microsoft SQL Server or Microsoft Commerce Server) is to read the preparation materials available for certification as a Microsoft Certified Solutions Developer (MCSD). Before you get defensive about not having enough time to get certified, or not having any interest in getting your certification (maybe you do—there *are* benefits, you know), let me just state that the point of the journey is not necessarily to arrive. In other words, you don't have to get your certification for the preparation materials to be useful; in fact, the materials might teach you things that you thought you knew well but actually didn't know as well as you thought you did. The fact of the matter is that the coursework and the requirements to get through the certification process are rigorous, difficult, and quite detail-oriented. If you have what it takes to get your certification, you have an extremely strong grasp of the fundamentals (and then some) of application programming and the developer-centric information about Windows platforms.

You are required to pass a set of core exams to get an MCSD certification, and then you must choose one topic from many available electives exams to complete your certification requirements. Core exams are chosen from among a group of available exams; you must pass a total of three exams to complete the core requirements. There are "tracks" that candidates generally choose which point their certification in a given direction, such as C++ development or Visual Basic development. The core exams and their exam numbers (at the time of publication) are as follows.

Desktop Applications Development (one required):

- Designing and Implementing Desktop Applications with Visual C++ 6.0 (70-016)
- Designing and Implementing Desktop Applications with Visual FoxPro 6.0 (70-156)
- Designing and Implementing Desktop Applications with Visual Basic 6.0 (70-176)

Distributed Applications Development (one required):

- Designing and Implementing Distributed Applications with Visual C++ 6.0 (70-015)
- Designing and Implementing Distributed Applications with Visual FoxPro 6.0 (70-155)
- Designing and Implementing Distributed Applications with Visual Basic 6.0 (70-175)

Solutions Architecture:

- Analyzing Requirements and Defining Solution Architectures (70-100)

Elective exams enable candidates to choose from a number of additional exams to complete their MCSD exam requirements. The following MCSD elective exams are available:

- Any Desktop or Distributed exam not used as a core requirement
- Designing and Implementing Data Warehouses with Microsoft SQL Server 7.0 (70-019)
- Developing Applications with C++ Using the Microsoft Foundation Class Library (70-024)
- Implementing OLE in Microsoft Foundation Class Applications (70-025)
- Implementing a Database Design on Microsoft SQL Server 6.5 (70-027)
- Designing and Implementing Databases with Microsoft SQL Server 7.0 (70-029)
- Designing and Implementing Web Sites with Microsoft FrontPage 98 (70-055)
- Designing and Implementing Commerce Solutions with Microsoft Site Server 3.0, Commerce Edition (70-057)
- Application Development with Microsoft Access for Windows 95 and the Microsoft Access Developer's Toolkit (70-069)
- Designing and Implementing Solutions with Microsoft Office 2000 and Microsoft Visual Basic for Applications (70-091)
- Designing and Implementing Database Applications with Microsoft Access 2000 (70-097)
- Designing and Implementing Collaborative Solutions with Microsoft Outlook 2000 and Microsoft Exchange Server 5.5 (70-105)
- Designing and Implementing Web Solutions with Microsoft Visual InterDev 6.0 (70-152)
- Developing Applications with Microsoft Visual Basic 5.0 (70-165)

The good news is that because there are exams you must pass to become certified, there are books and other material out there to teach you how to meet the knowledge level necessary to pass the exams. That means those resources are available to you—regardless of whether you care about becoming an MCSD.

The way to leverage this information is to get study materials for one or more of these exams and go through the exam preparation material (don't be fooled by believing that if the book is bigger, it must be better, because that certainly isn't always the case.) Exam preparation material is available from such publishers as Microsoft Press, IDG, Sybex, and others. Most exam preparation texts also have practice exams that let you assess your grasp on the material. You might be surprised how much you learn, even though you may have been in the field working on complex projects for some time.

Exam requirements, as well as the exams themselves, can change over time; more electives become available, exams based on previous versions of software are retired, and so on. You should check the status of individual exams (such as whether one of the exams listed has been retired) before moving forward with your certification plans. For more information about the certification process, or for more information about the exams, check out Microsoft's certification web site at *www.microsoft.com/train_cert/dev*.

Conferences

Like any industry, Microsoft and the development industry as a whole sponsor conferences on various topics throughout the year and around the world. There are probably more conferences available than any one human could possibly attend and still maintain his or her sanity, but often a given conference is geared toward a focused topic, so choosing to focus on a particular development topic enables developers to winnow the number of conferences that apply to their efforts and interests.

MSDN itself hosts or sponsors almost one hundred conferences a year (some of them are regional, and duplicated in different locations, so these could be considered one conference that happens multiple times). Other conferences are held in one central location, such as the big one—the Professional Developers Conference (PDC). Regardless of which conference you're looking for, Microsoft has provided a central site for event information, enabling users to search the site for conferences, based on many different criteria. To find out what conferences or other events are going on in your area of interest of development, go to *events.microsoft.com*.

Other Resources

Other resources are available for developers of Windows applications, some of which might be mainstays for one developer and unheard of for another. The list of developer resources in this chapter has been geared toward getting you more than started with finding the developer resources you need; it's geared toward getting you 100 percent of the way, but there are always exceptions.

Perhaps you're just getting started and you want more hands-on instruction than MSDN Online or MCSD preparation materials provide. Where can you go? One option is to check out your local college for instructor-led courses. Most community colleges offer night classes, and increasingly, community colleges are outfitted with pretty nice computer labs that enable you to get hands-on development instruction and experience without having to work on a 386/20.

There are undoubtedly other resources that some people know about that have been useful, or maybe invaluable. If you know of a resource that should be shared, send me e-mail at *winprs@microsoft.com*, and who knows—maybe someone else will benefit from your knowledge.

If you're sending mail about a particularly useful resource, simply put "Resources" in the subject line. There aren't any guarantees that you'll get a reply, but I'll read all of the mail and do what I can to ensure that your resource idea gets considered.

CHAPTER 5

Understanding Remote Access Transmission Technologies

First things first: This chapter is not absolutely necessary to develop programs that make use of RAS or the remote access capabilities of RRAS. But then again, you don't need an electrician's certification to play with circuit breakers and wiring (but the knowledge that goes along with that certification can really help). The point is that garnering a knowledge base for the technologies associated with your work (whether it's developing, plumbing, or electrical work) can go a long way in helping you better *understand* what you're working with.

This chapter is geared toward providing you with a quick, reasonably concise explanation of the transmission technologies associated with remote access. Of course, there are likely going to be improvements and changes to these technologies as time marches on, but from this basis, you will be better prepared to understand those changes. How can this chapter help you develop better applications? If you're doing the debugging or testing of your applications, these can help tremendously; is it your application that isn't working properly (or providing the best performance), or is it the fact that your 56k modem is actually achieving only 33kbps (because noisy lines can make modems drop in their transmission rates to achieve a reliable connection)? Is ATM a telephone technology for Regional Bell Operating Companies (RBOCs), or is it a network transmission technology (see the ATM section later in this chapter)? Knowing the answers to questions such as these (and many more) can make you a more savvy remote access developer, and might even expand your knowledge base a bit, which is a worthy achievement in its own right.

Analog Modem Technology

Analog modems make up the bulk of the long-distance computer communications devices today. These modems are analog because we live in a world (or a time) where the most widely available network—the telecommunications network—is based almost entirely on analog connectivity for the end user (this is changing with ADSL and Cable Modems, but it has a long way to go).

As most of you know, however, computers don't function natively on analog signals; when the computer is communicating internally between subsystems, such as when the hard drive is loading an application into memory, it doesn't communicate with analog data. The computer instead functions on digital data (ones and zeros), requiring some means of converting the digital information on a computer into a form that can be transmitted over the analog device connected to the telephone network. The devices used to achieve this transition are today's analog modems.

What is a modem? Most people can point one out in a lineup, but actually answering what a modem is, what it does, how it does what it does, and where it got its name are generally not too clear. Quick answers: Modems are communications devices that use the telecommunications network to transmit data over geographically distant sites. Modems take digital signals (from the computer) and turn them into analog signals that can be transmitted over the telephone network. Modems do this by taking the serial data they receive from a COM port and translating it into specific analog signals that can be understood by the modem on the other side of the connection and translated back into digital data. Modem stands for MOdulator/DEModulator.

Getting Data to the Modem

It's late at night and you have a sales information file on your hard drive at home (which is where you are for this illustration), which absolutely needs to be on the company server before the day ends. That means you need to get the sales information file from your computer at home to the server at work, and that's going to be achieved when you make the connection between your modem and one of the modems hanging off the RRAS Server at corporate headquarters. What does that mean? It means that you need to get the data from your hard drive to the modem, then the modem needs to send it over the telecommunications network (PSTN, from here forward) to the corporate modem, which will then forward it appropriately to the server to which you're trying to transfer the file. Nothing to it.

Parallel Versus Serial Communication

A computer communicates in parallel communication, which means that it sends multiple bits of data at once. Serial communication, in contrast, sends one bit at a time. Think of it this way: When you go to Disneyland or one of the Six Flags theme parks, at the front gate there are a number of turnstiles that funnel incoming guests through the ticket gate. If there are eight turnstiles, when looked at as a whole, they are letting people through the gates eight people at a time. So you have eight lines, and if the ticket-takers are synchronized (taking tickets at precisely the same time), people would be entering the park eight people at a time.

Serial communication, in this example, would be like a theme park that had only one turnstile, and thus could let people come through the gate only one at a time.

Because your computer's bus utilizes parallel communication to send data wherever it's going around the computer, and the communications port needs to communicate serially, there must be a means by which the computer's parallel means of communication and the serial port's serial means of communication are translated for one another. This is done by a chip called a Universal Asynchronous Receiver/Transmitter (UART). Each communications (COM) port on your computer also has a UART associated with it, which translates parallel data to serial data, and vice versa. Figure 5-1 illustrates the mechanism.

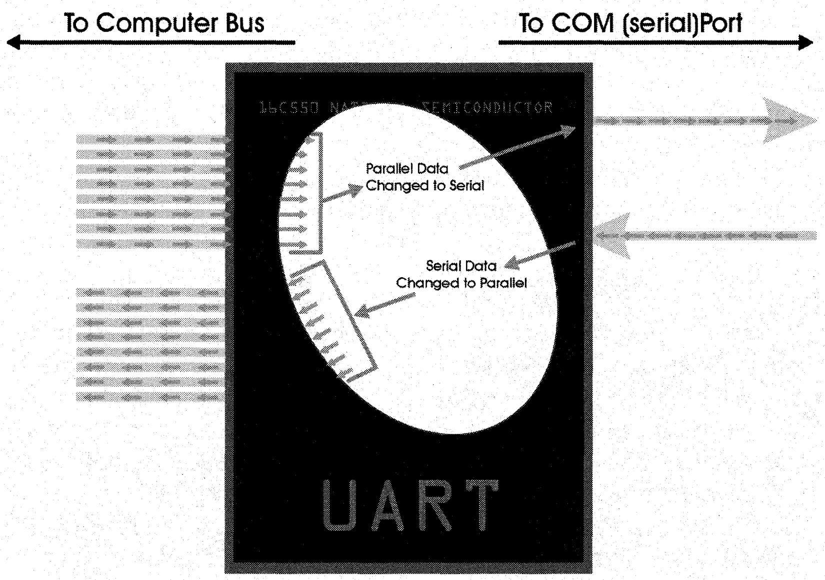


Figure 5-1: Parallel Data Going Through the UART to the Communications Port, and Vice Versa.

Part of the responsibility of the UART is to add (upon transmission to the COM port) and remove (upon reception from the COM port) start and stop bits that "frame" serial data to let the receiver of the data know when the beginning and end of a byte's data is reached. In very loose terms, a start bit is similar to the capitalization we do at the beginning of a sentence and the stop bit is similar to the period we put at the end of a sentence. Without either, it would be much more difficult for us to determine where a sentence starts and where one ends; though we may be able to figure it out from context, there is still room for ambiguity and computers don't have much tolerance for such inaccuracies. The start and stop bits in a sentence let us read more clearly, understand much better, and are universally (in English, at least) accepted delimiters. The same can be said for start and stop bits in serial communication.

All UARTs are not created equal. Some are better than others, some are faster than others, and some still are faster and better than all the others—and of course, generally more expensive. Internal modems use a UART that comes built into the modem card itself and thus doesn't utilize the UART (or COM port) built into your computer.

Whether this is good news or not is largely a matter of opinion, but know that you are at the mercy of the modem manufacturer for the quality of your internal modem's UART, which means that there is another performance parameter to take into consideration when purchasing your internal modem (if you do such a thing—I wouldn't suggest it). For the rest of this example, I'll presume you have an external modem.

Back to the sales file. You've instructed your computer to send it to the corporate LAN (I'm presuming at this point that you're connected to the corporate LAN using your modem) and now the UART on your computer is taking the data that the hard drive is sending over the bus (the computer's internal freeway system for its data) and translating it from the computer's native parallel form of communication to serial communication—in other words, to the COM port. Your modem, then, is connected to your COM port and is accepting the serial data from the COM port. Your modem takes this serially transmitted data, looks at it, maybe compresses it if that's part of its functionality/feature set, and then sends it out over the wire and across the PSTN. This, of course, is where the modem's laws of physics kick in.

How Analog Modems Operate

How were all of these analog transmissions achieved, once this data got to the modem? How did modem makers go from 300bps—that's bits, not bytes, per second—to almost 200 times that throughput? It all has to do with the means by which they modulated the data, and an explanation of how such modulation is done requires more than a cursory, narrative description of how modems came into being and took their permanent place at the PC dinner table. That explanation requires an overall view of the means by which modulation is achieved, the constraints under which modems must function (dictated by the PSTN), and the means by which these conditions are married in today's contemporary modems. In short, such an explanation requires details. Only after these details are fleshed out will we see why the bandwidth available for the analog modem is running out.

PSTN Bandwidth

I've already mentioned the confines within which modems connect to the PSTN, but here we're going to look a little closer and throw in a couple of illustrations to show just where these lines are drawn.

Remember that the PSTN was built around the need to provide voice service to the masses. Two terms are important in that statement: "voice" and "masses." When the infrastructure of the PSTN was being created and the decision regarding how much bandwidth to provide was made (and where that bandwidth was on the analog spectrum), economics ruled, as they probably should have. A certain level of quality was necessary, but to provide crystal-clear voice transmissions wasn't the goal; the goal was to provide voice service to the masses, and that meant having the capability to transmit massive amounts of calls over the PSTN infrastructure all at once, or at least a lot of them at once.

It was found that the range of analog signals necessary for transmission of intelligible, reasonably clear communicated speech could be constrained to the range of 300 to 3300 Hz. Such constraint provided for an economy in the transmission of lots of these voices because the more range provided, the more bandwidth necessary for transmission over the PSTN infrastructure. The range over which the transmission of voice was allowed to transpire was set, and the equipment that handled voice traffic (switches) filtered everything below and above this set range. The result was an available range similar to what you see in Figure 5-2.

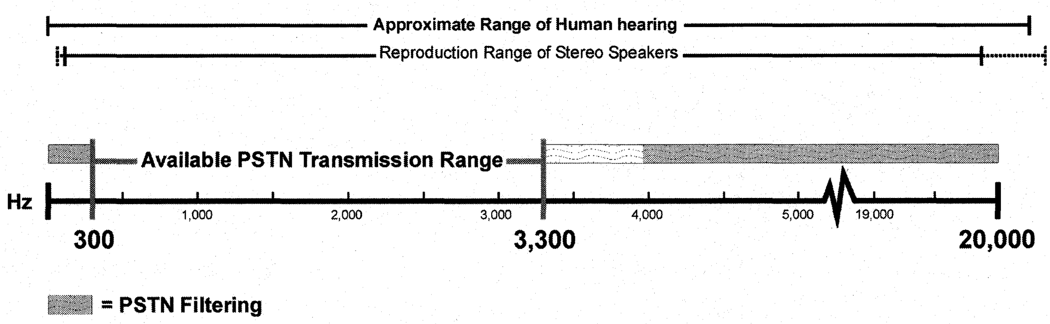


Figure 5-2: The Range Available for Voice Transmission Over the PSTN.

This worked great for the telecommunications network—it could limit the amount of resources necessary to transmit voice from one end of the network to the other—but the same factor that provided an economy of transmission for the telecommunications network also bridled the amount of theoretical bandwidth available for the transmission of data. In short, PSTN filtering limits the amount of bandwidth available to analog modems.

Modulation

Modulation is the conversion of digital data into tones, or analog form. For example, when you put the receiver to your ear and hit a number key on the telephone, a certain and specific tone is emitted for as long as you hold down that number. On the receiving side of things (the PSTN), the tone is recognized as the representation of a number (the number of the key you pressed), and as those numbers are pressed, the phone company knows that your intent is to connect to another telephone and it puts the signal through. Modems do much the same thing, except that the tones that they emit are going much, much faster than your finger can press, and the means by which they generate these tones (as well as the way the present certain characteristics of the tone, such as its phase) follow a set of standards that all modems understand and translate into data. The three major means by which analog modems throughout their existence have transmitted their data are Frequency Modulation (FM), Amplitude Modulation (AM), and Phase Modulation (PM) or Phase Shift Keying (PSK).

A word of warning: A true, complete, and in-depth technical treatment of modem modulation is the subject of an entire book, not one chapter's section. I'll give you a good overview and enough information and explanation to help you understand how it works.

If you want more specifics, more mathematical depth, or just more bulk, there are technical papers and telecommunication tomes in abundance that can articulate what equations were used to get where we are today. This is the hands-on version; the "pencils-on" and "calculators-on" versions can be found at your local university bookstore in the "in-depth treatment of technical telecommunications theories" section.

Modulation methods

FM, utilized in early modems, implements a changing of the carrier frequency (i.e., the analog signal, or sine wave) to represent the value of the bit being transmitted. For example, if we use a frequency of 1070Hz to represent zero, and a frequency of 1270Hz to represent one (remember that we must operate in the 300Hz to 3300Hz range or the PSTN will filter us out), then we can switch between these two frequencies and transfer data. If the baud rate is 300, which means that the signal can change 300 times per second, and with each change we communicate one bit of information, then the bit rate is equivalent to the baud. If this is the case, then we also know—because a baud rate of 300 means that there are 300 signal changes per second—that the duration of one of these little pieces of information is 1/300th of a second. Figure 5-3 presents a visual representation of FM.

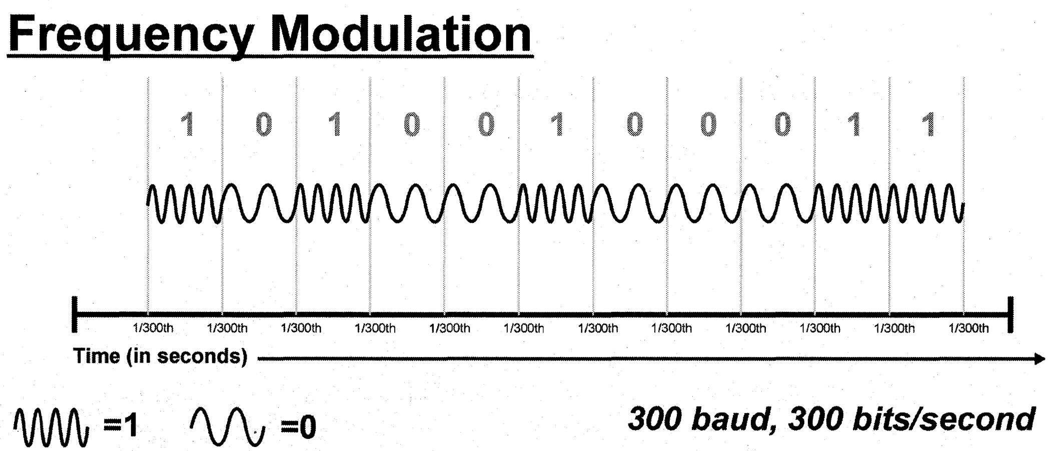


Figure 5-3: Frequency Modulation.

Though FM was used in some modems that operated at higher speeds than 300bps, in contemporary modems FM has been replaced by a combination of the two following approaches.

AM achieves the transmission of zeros and ones through the use of a change in the amplitude, or height, of the analog signal. Figure 5-4 puts this into a picture.

As you can see from Figure 5-4, AM uses a change in the amplitude of the analog signal carrying its information to create the distinction between ones and zeros. If AM is straightforward and easy to comprehend, then the third means to impress intelligence on an analog signal (or in simpler terms, to transmit data using an analog signal), PM, makes up for it.

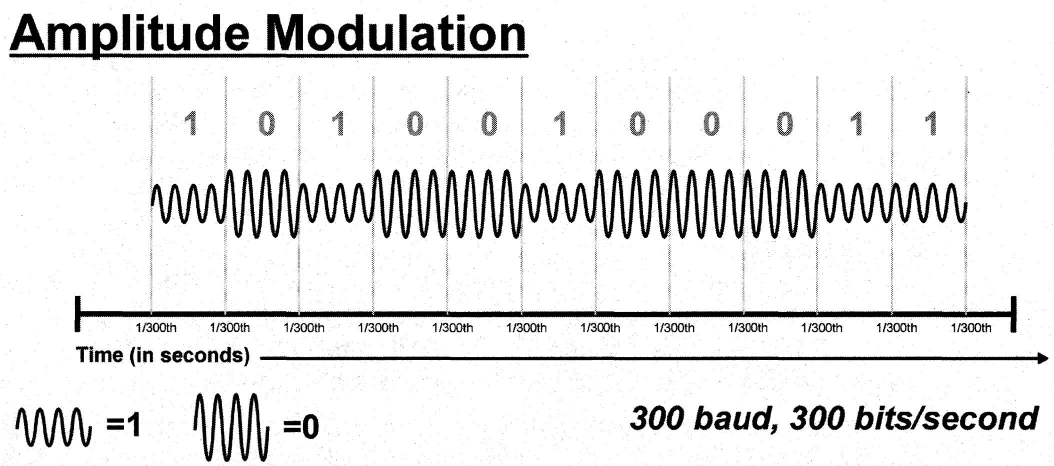


Figure 5-4: Amplitude Modulation.

PM changes the phase of the sine wave being transmitted in order to represent a value. PM is certainly best explained with a picture (Figure 5-5) and I think a thousand words is on the low side.

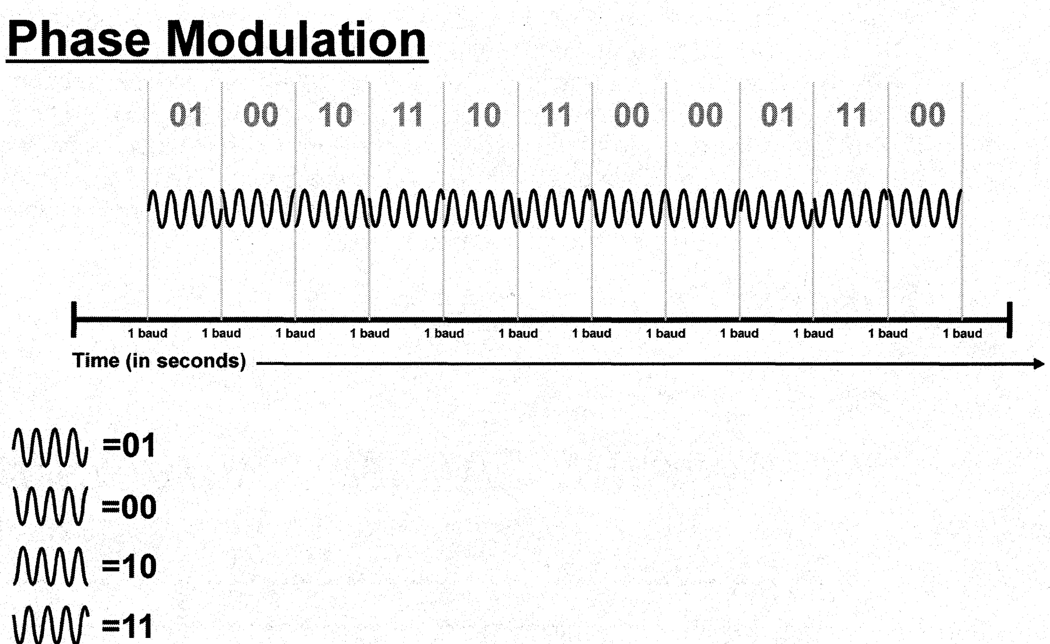


Figure 5-5: Phase Modulation.

PM is reserved for more sophisticated modems, generally those that are 14,400bps and above, which puts most (or all) of our contemporary modems in the category of PM, also knows as PSK. As mentioned earlier, AM and PM are often combined, the result of which is that a single signal can represent more than one bit. More on that later. First, let's take a look at what a simple modulation scenario, using a modem that implements FM, might shake out to be.

Simple modulation

In terms of implementation and explanation, the simple version of modulation comes when the bit rate is equal to the baud. This hasn't been the mode of operation since early in the 2400bps/baud modem days, but it is the building block of more complex modulation methods, so the discussion appropriately starts here.

We already know what bits per second is—the amount of bits (ones or zeros) that can be transmitted in a given second—but we haven't really clarified baud. This is a perfect point at which to clarify.

The term baud is defined as the number of signal changes for an analog transmission in a given second. In early modems, the baud rate was equal to the bits per second rate, because each change in the analog signal (actually a sine wave) represented one bit. From our earlier explanation of FM, we were talking about a modem that transmitted zeros with a signal setting of 1070Hz, and transmitted ones with a signal setting of 1270Hz. In the simple version of modulation, a zero will be communicated across the PSTN by transmitting the carrier signal at 1070Hz for the duration allotted once signal value (that's 1/300th of a second on a 300 baud modem). If the next bit to be transmitted is also a zero, the carrier signal will continue at the frequency that corresponds to zero (1070Hz) for the duration of another signal value (another 1/300th of a second). This is very straightforward, and as Figure 5-6 suggests, the mapping of single bit to each signal change makes for an easily understood scenario.

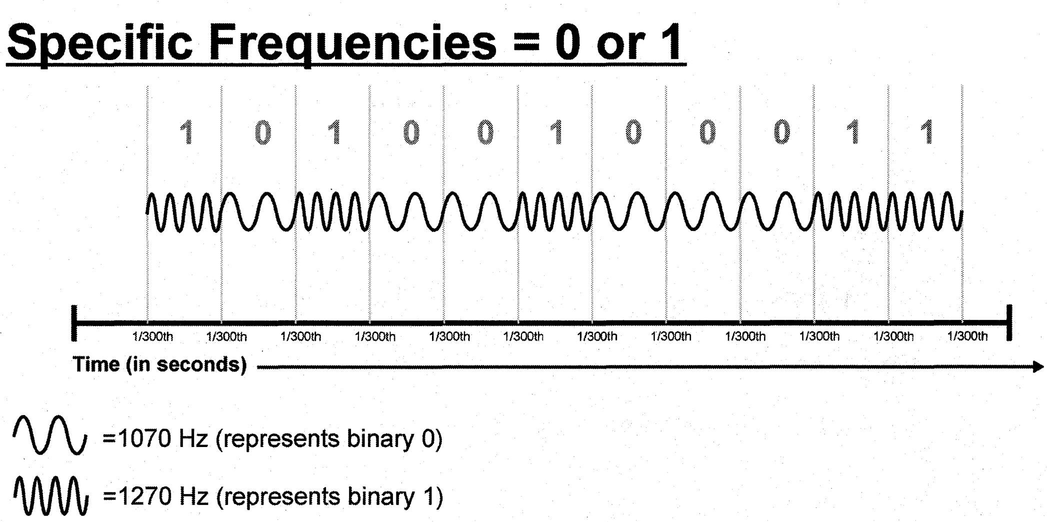


Figure 5-6: Mapping One Bit for Each Baud.

But in real life, things are rarely that simple. The next section explains the more complex version of analog modem modulation.

Complex modulation

To respond to the increasing need for speed, modem manufacturers developed a way to transmit more data through the same (finite) baud rate available through the analog PSTN. In other words, they figured out how to transmit more than one bit for each signal change. Of course there were some acronyms that evolved from such ideas; dibit encoding deals with the encoding of two bits per signal change, tribit encoding involves allowing each signal change to represent three bits (imagine that). There's also the venerable QAM, or Quadrature Amplitude Modulation, which results in four bits being transmitted for each signal change (QAM32 and QAM64 are variations of QAM that represent five and six bits respectively). These days, however, bit transmission rates are in the nine bits per signal change neighborhood for analog modems.

This mapping of more than one bit per signal change is generally achieved through the combination of PM or PSK (PM/PSK) and AM. But a little bit of explanation is required for a reasonable amount of understanding to be achieved with regard to how this mapping is done; it isn't quite as straight-forward as it sounds at first pass. QAM, with its four bits per signal change operation, requires 16 distinct states. 16? Yes, because as you remember, we're talking about the representation of binary information, and for all possible combinations of four binary bits to be provided, there must be 16 states. Look at Figure 5-7.

As you increase the amount of bits you want to transmit with each signal change, the number of discreet states that must be available to represent every available bit combination grows in a binary fashion. In other words, the amount of discreet states that must be represented doubles every time you add one bit to the number of bits you're trying to transmit with each signal change. With QAM, which represents four bits per signal change, you need 16 (24) discreet states; if you want to send five bits per signal change, you need 32 (25) discreet states. Once you get to 9 bits, you need 512 (29) discreet states. That's a lot of states; too many to transmit on conventional PSTN lines without lots of errors, actually. As the states increase, the difference (in amplitude or phase shift) between the states becomes smaller and smaller, until you get to the point where the state changes are so susceptible to noise in the line—noise that can attenuate the signal and thus make it appear changed once it gets to the receiver—that the transmission becomes error laden to the point that the use of so many signal states becomes its own worst enemy. A better way to manage errors was needed, and in order to get past the 14,400bps modem speed with any consistency, it was absolutely necessary. That better way came in the form of Trellis Coding.

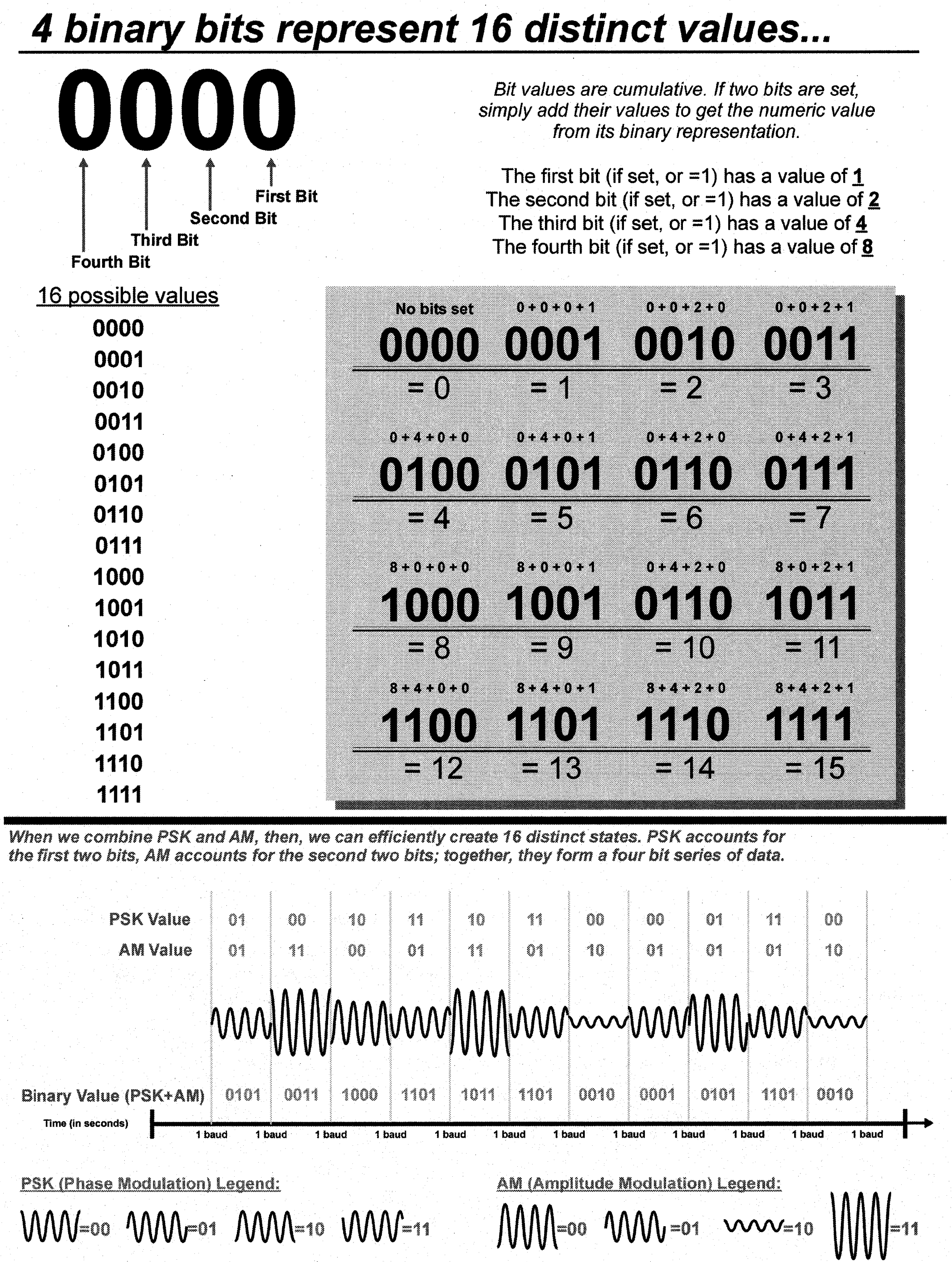


Figure 5-7: Four Binary Bits Being Represented as 16 Changes.

Trellis Coding is a method of encoding data that is much more robust than conventional QAM encoding; Trellis Coding can tolerate more than twice as much noise or other line imperfections as QAM modems, and its sophisticated error detection techniques reduces the likelihood of transmission errors by orders of magnitude. The decrease in errors is achieved by adding redundancy into the bit stream that essentially "steers" the interpretation of the received signal to the correct value. In Trellis Coding, only certain sequences of ones and zeros are valid. As the data stream passes through the Trellis coding logic, its bit sequences (zeros and ones) are evaluated and then impressed with redundant bits and sent over the wire. When the transmission reaches its intended receiver, the value is sent back through the receiving modem's Trellis Coding logic and checked for "Trellis Coding" validity, and then handled appropriately (rejected or passed).

Today's modems transmitting at 33.6kbps use Trellis Coding and other techniques such as equalization (built-in components that compensate for channel distortion) to get all sorts of data across the wire. We've hit the major points here.

The bottom line, then, is that contemporary modems take the synchronous bits (received through the COM port via the UART, which has converted the parallel data from the computer into serial data that the modem can understand) and group them into multiple bit groups (often groups of nine), then represent that group of bits by choosing the unique signal state (through a combination of AM and PM) that corresponds to that specific group of bits, then transmits that unique signal (and thus the bit group) within a single cycle. To avoid errors—errors that would certainly result from the closely aligned constellation patterns of today's modems—contemporary modems use the complex mathematical formulas and encoding logic incorporated in Trellis Coding to greatly reduce the chance of mistaken signal state identity.

The 56k version

56k technology takes even the reduced Trellis Coding mistaken signal-state identity problems to task. This analog modem technology, which is actually a hybrid digital-analog technology, comes in the form of a touted but mostly untrue 56,000bps downstream throughput technology commonly referred to as 56k technology.

56k technology is based on the fact that most remote access service providers, such as an MSN or an ISP, or many mid-sized or larger corporate remote access facilities, have digital T1 connections servicing their modem banks. This digital connection, the T1 servicing the server modems, is critical to the implementation of 56k technology, for it ensures that only one digital to analog conversion will occur in the path between the server modem and the client modem and removes all errors associated with degraded or distorted analog signals on the server modem's loop. Notice here that we can specify, or define, server modem and client modem: The server modem is the 56k modem that is directly attached to the digital T1 facility, while the client modem is the 56k modem that is attached to the user's standard analog subscriber loop. Notice also that both modems are 56k modems, which is a prerequisite for establishing 56k connections. Figure 5-8 shows how the physical setup of this configuration would look in terms of the elements involved in creating the connection between modems.

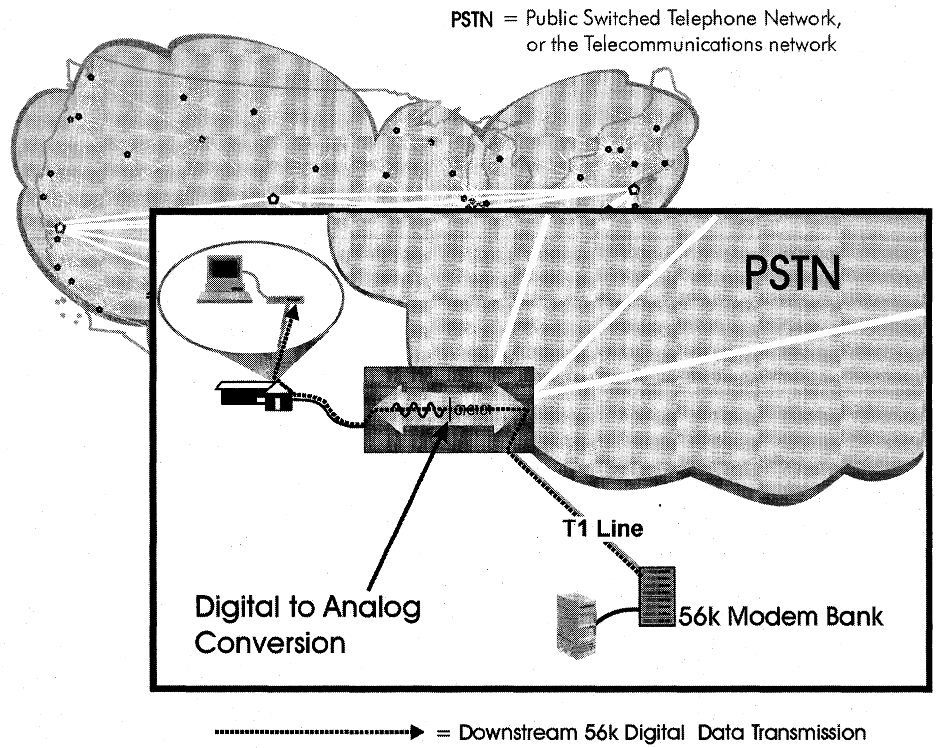


Figure 5-8: The PSTN, with a Remote Access Provider Connected to a T1 and a Client Connected to the Telephone Line.

The effects of this isolation of digital to analog conversions provides the means by which 56k technology can be implemented. The PSTN digitizes transmissions when coming from the analog subscriber loop, or in more widely used terms, does an analog-to-digital conversion of the data. The information travels across the digital core PSTN until it reaches the Central Office (CO) servicing the receiving connection's subscriber loop, where the digital information that has traversed the PSTN is turned back into analog form.

56k technology removes the initial analog-to-digital conversion, creating a communication between modems that contains only one conversion—that which occurs as the digital information sent from the server's 56k modem reaches the client modem's subscriber loop, where it is put into analog form. There are 256 possible representations of analog information (8-bits per sample, which in binary creates 256 possible representations); the 56k server modem uses that knowledge to its advantage by transmitting those specific codes. By avoiding the analog-to-digital conversion, and thus avoiding the Analog to Digital Converter's (ADC) interpretation of analog signals that may have been distorted or attenuated by line noise, server modems equipped with 56k technology can transmit the binary representations of the analog signals, thus avoiding all errors associated with the first analog loop.

Although not all of the 256 representations can be utilized, largely because as they approach 0Hz the space between those analog representations of digital data is too small and thus too prone to errors with even the smallest line noise (line noise is still an issue on the client modem's subscriber loop), many of the 256 representations can be utilized and discretely transmitted to and through the Digital to Analog Converter (DAC) to the client. It is digital all the way to the client subscriber loop DAC, and due to that fact, errors and limitations resulting from what would be the server's analog-to-digital conversion are removed, allowing throughput levels that approach the absolute ceiling of DS0s (Digital Signal level 0), the individual analog line payload, 64kbps.

ISDN Technology

First a quick disclaimer: ISDN technology is not analog modem technology, but its discussion as a (waning) client-end transmission technology meant its discussion fit better here (in this section) than in any of the others. Disclaimer complete.

You've probably heard of ISDN, and depending on whom you were listening to, probably heard how it will never get off the ground, or how it's noticeably way, way faster than even 56k technology. In remote access connectivity reality, both may be correct. The technology has drawn praise and prejudice, and along the way pro tem meanings have been coined to usurp the official acronym, such as ISDN standing for "I Still Don't Know." Whoever thought that one up obviously wasn't a spelling bee champion.

ISDN actually stands for Integrated Services Digital Network, and its most notable technological departure from today's analog modems is easily explained: ISDN removes the analog part of the data transmission process. There is no analog local loop; it is instead all digital, and that digital connection enables users to achieve the full 64kbps per channel that T-Carriers and ISDN PRI frames offer to each channel every 125 microseconds, thanks in part to its technological design that puts signaling and administration features out-of-band. I've loaded this paragraph with plenty of unexplained technology tidbits; let's get to the explanation part of it.

First, the all-digital part. Telephone lines are analog, so they can take your voice (an analog signal) in its native format (as you talk into the phone) and do their conversion from its natural analog form into digital form so that the PSTN can send the representation of your voice over the PSTN infrastructure in an efficient way. With ISDN, you call up your local telephone company and say, "I'd like ISDN." If ISDN service is available in your area, and once all the details of getting the service are ironed out, your telephone line (or your extra line) is physically changed at your CO to an ISDN interface, and in effect, the analog-to-digital converter that is on everyone else's line is removed. To clarify this, take a look at Figure 5-9.

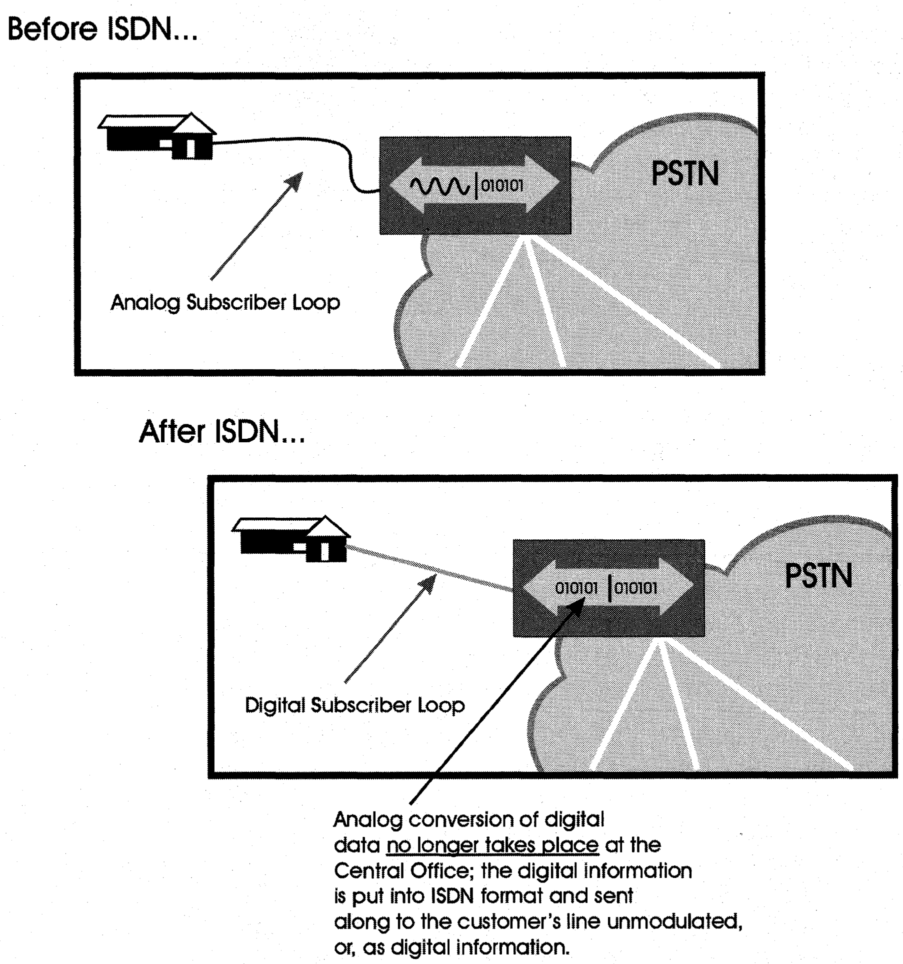


Figure 5-9: An Original Analog Line Getting Changed to an ISDN Line.

On to the second point: ISDN provides the ability to utilize the entire 64kbps channel provided by the T-Carrier/ISDN PRI standard. Explanation of this brings us back to the digital end-to-end characteristic of ISDN. Through such digital implementation, the need to interpret analog information and translate that analog information into its binary representation is removed. What that leaves, then, is the ability to use all 8-bits of the per-cycle sample for data (still accomplished at 8,000 samples per second, or in 125*s intervals. This is a trend you will see throughout telecommunications and its new technologies). 8,000 samples * 8-bits per sample = 64,000 bits per second, per channel.

ISDN comes in two standard interfaces: Basic Rate Interface (BRI) and Primary Rate Interface (PRI).

BRI is geared more for the end user or small company, and its standard offering is two bearer or B Channels, each operating at 64kbps as explained earlier, and one data or D channel. This configuration is often referred to as 2B+D. The B channels handle the data (or voice, video transmission, or whatever other ISDN-featured technology you want to use the channel for), while the D channel manages the administrative part of the ISDN service suite.

PRI is geared toward the remote access service center, or PBX, or other multi-line (the term is being used loosely here) services planned for the ISDN PRI interface. PRI is similar in use to a T1 in that it can support multiple remote access sessions through one interface. PRI also has a bandwidth of 1.536Mbps, as does a T1, but its division is slightly different than that of a T1. The PRI is generally divided (in remote access solutions) into 23 64k B channels and one D channel. Thus the PRI is the ISDN interface that would be used at the corporate RRAS/remote access site to provide ISDN connectivity to remote users.

ISDN is more complex than conventional modems. Its implementation requires more patience, especially since you can't just plug into your existing phone line and call it good. And with the availability of ADSL or Cable Modems these days (both of which provide much more bandwidth), it's a hard sell to just about anyone.

Residential Broadband Technology

ADSL Technology

What's one of the biggest financial assets of the telecommunications industry? The existing wiring plant; all those pairs of twisted wire running under the ground of almost every street in America, bringing a dial tone to anyone who wants it and everyone who needs it. It's everywhere, and it's a huge asset. It's in everyone's home, everyone's business, and in many places, there's more than one pair of wires to each residence. That's a lot of contact with a lot of people, and those people want a lot of bandwidth for a lot of different reasons. The telephone companies want to provide that bandwidth, however it gets to you. What's perhaps the best way to do so? Well, I suspect using an existing, omni-present, already-paid-for telephone wiring infrastructure would be a good means of providing high data-rate services, at least as far as the local telephone companies are concerned. The problem lies with their wiring: Standard twisted pair wiring, the kind that everyone has in their house, was meant for voice, not high-speed data services. The telephone companies thought (and said to Bell Labs, now Lucent), "what if we could use that existing wiring infrastructure to provide high speed data services?" Enter ADSL.

ADSL stands for Asymmetric Digital Subscriber Line, and its technology is the result of a search to find a way to utilize existing copper twisted-pair wiring—standard phone lines—to provide a high bandwidth solution. It has had a few (too many) acronyms, including HDSL, SDSL, RADSL, VADSL and VDSL, and although some of these acronyms stand for differing applications of the overall DSL technology, ADSL has

emerged the most widely known, likely because it's the most used application of xDSL. The reason for its wide use (in terms of xDSL technologies) is that ADSL's distance requirements encompass the majority of existing telephone lines, and because it has the potential to be deployed in large volume in the near future. VDSL is actually a higher-throughput rate of ADSL; differentiation and explanations of the differences between them and the other xDSL technologies will be covered in the following section.

ADSL Technology Overview

As the term "Asymmetric" suggests, ADSL technology provides different throughput levels for each direction, or in more direct terms, ADSL can pull data downstream at a much higher throughput rate than it can send data upstream.

ADSL's theory is relatively straightforward in its explanation: Through the use of a modem pair, one at the customer premise and one at the local CO, data is transferred at very high speeds to the customer premise equipment (downstream). A lower level of bandwidth is afforded for the upstream communication, but the ratio is very much in the downstream favor (for example, 756kbps downstream : 128kbps upstream), which coincides perfectly with the way people use their residential services. Audio and/or video content such as movies (incoming, or downstream), Internet access (Web page viewing is mostly downstream), and radio (downstream, and reproduced with great clarity if it's digital) are all downstream content deliverables, and these are just some of the more obvious examples.

One of the most attractive aspects of ADSL is the fact that it incorporates the use of your existing telephone line into its technology, which means you need only one telephone line to keep your existing unmodified (as far as you can tell) telephone service, and you get the full range of bandwidth associated with ADSL without any conflict. People in the household can be on the telephone, surfing the Web, listening to some heavy metal radio station, watching some on-demand movie, and playing an interactive Internet-based Quake Arena deathmatch, all at the same time with ADSL. No more obnoxious data signals or interrupted transfers because someone picked up the phone when you were getting a fax or sending e-mail. The setup of ADSL in the home looks similar to Figure 5-10.

ADSL's technical implementation is somewhat more complex, but because we've already been through analog modem explanations, it will be much easier to explain and understand.

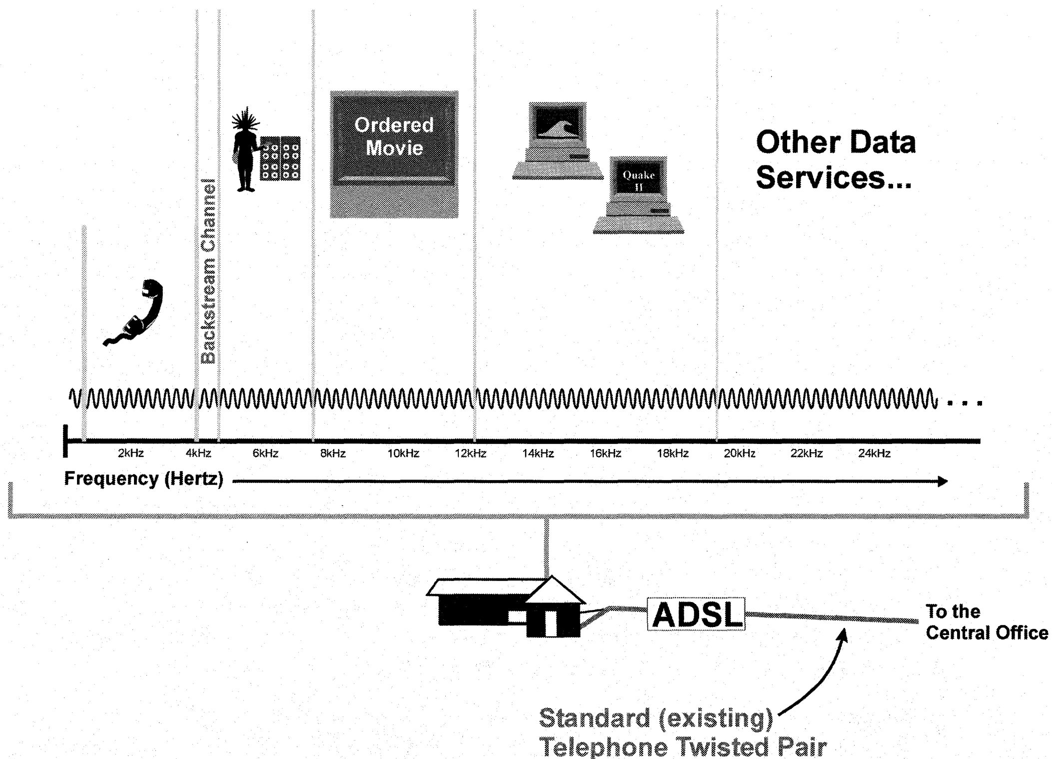


Figure 5-10: ADSL in Simultaneous Use in the Home.

We remember that standard telephone service uses the 300Hz to 3300Hz frequency range for telephone calls; for more practical reading, we'll simply say that standard telephone services uses frequencies between 0kHz and 4kHz (0Hz and 4000Hz—some of the frequency spectrum above 3.3khz is used for administrative purposes). As stated previously, data within that spectrum is digitized and passed along the circuit toward its destination and any signal or information above that range is filtered out at the local CO. This is why standard analog modems have such constraints under which they must operate, since all the data they want to transmit must be sent within that frequency range, as it otherwise would be filtered out and never reach its destination. I remember thinking, "why don't they just do away with that filtering and allow for more use of the frequency spectrum?" Guess what: ADSL technology does away with that filtering.

By removing the constraints of the standard PSTN filtering, ADSL can appropriately divide the resulting available spectrum among standard telephone service, data service, video-on-demand service, radio service, and whatever other services come along.

With all this high-throughput talk, there is one very important consideration to keep in mind when touting the benefits of ADSL or any xDSL technology: It is distance-oriented, and the greater the distance between the residence and CO, the lower your maximum throughput. There are limits to the frequency at which ADSL or any xDSL modem can operate due to attenuation and other physical characteristics that degrade the signal as it travels down the twisted pair wire. These signal losses or distortions, and their aggregated effects on the signal being transmitted between modems, result in the distance limitations placed on throughput capabilities of ADSL. Longer wire runs mean more signal distortion or loss, and as the wire gets shortened these effects are minimized, leaving more frequency range available for transmitting data. There are some ADSL implementations that attempt to adapt to imperfections in the attached telephone line, such as Rate-Adaptive ADSL (RADSL), which tests the line for noise or transmission impairments and adjusts its transmission to get the most throughput possible out of the available line quality (a great advantage of ADSL technology, IMHO). Despite this adaptation to the noise inherent with telephone lines, ADSL is still sensitive to line distances; the shorter the distance the better. Thus, shorter distances provide greater available bandwidth, and that discussion brings us to VDSL.

VDSL or Very high data rate Digital Subscriber Line, sometimes called VADSL when "Asymmetric" is thrown into that line of words, can be called the short, stocky cousin of ADSL. In short (excuse the pun), VDSL is a very high-speed version of ADSL. Though sometimes called VADSL, it is inaccurate at this early stage in the game to presume that VDSL will be asymmetric. Indeed, though maximum line lengths would be compromised in the process, it is possible that customers who would need the extremely high VDSL data rates would want (and get) symmetric service; in other words, those customers would want the high throughput in both directions of the connection.

The question then becomes: How does ADSL get all that information from one modem to the other? There are two technological camps with regard to which method is better, and those are CAP and DMT.

CAP stands for Carrierless Amplitude/Phase modulation, and is essentially a variant of QAM, which was discussed earlier in this chapter as a means of representing multiple bits with one signal change. CAP was not in the ANSI T1.413 standardization for ADSL technology, but success in its implementation in some field trials have reportedly resulted in some big name manufacturers of ADSL equipment lobbying for its inclusion in the ANSI standard.

DMT stands for Discrete Multi-Tone. DMT (in general terms) effectively divides the available frequency spectrum into discrete frequency segments, each of which (or many of which, for certain segments) is specifically allocated in its ADSL application for certain uses such as video channels, ISDN channels, or administrative signaling, which also reserves existing frequency ranges for standard telephone service. Often these segments are called channels. DMT is included as the standardized ADSL transmission technology in the ANSI T1.413 recommendation.

ADSL technology, though somewhat easy to explain in its theoretical and implementation approaches, is certainly not a simple feat of engineering; it is a genius of invention and implementation in its hardware and the algorithms that go into the innards of an ADSL modem, and we're fortunate enough take be able to take it for granted.

Cable Modem Technology

Cable companies also have a very large, very valuable installed infrastructure base, though it differs widely from the installed telecommunications base, as we'll investigate further when we discuss the technology behind cable modems. One differentiating feature of cable modem technology, however, is that cable modem technology has lots of potential bandwidth on which it can operate. How much? More than a T3. More than your Fast Ethernet can handle. More than an OC-3. More than an STS-5. Lots. But if we lived in a world where it was all that easy, we'd all have cable modems, there would be no such thing as bottlenecks, and money would grow on trees (at least in my yard).

Cable Modem Technology Overview

Cable modem functionality requires a quick overview of the way CATV operates, and the means by which we get all those nifty, never-watched cable channels piped into our living rooms.

CATV technology creates individual channels through the use of Frequency Division Multiplexing, or FDM, by dividing the available frequency spectrum of the well-shielded CATV coax cable into 6Mhz segments. These segments, more appropriately called channels, are used to transmit (broadcast) one-way information out to the attached nodes. Those nodes, connected in a branching tree (or tree and branch) topology, have certain tuners attached to them that allow them to focus on a particular 6Mhz channel and transmit the information they receive onto some medium (often a television).

Cable Modem technology, then, utilizes a 6Mhz channel that has been reserved for receiving data; current downstream rates are either approximately 10Mbps or 36Mbps, depending on whether QAM64 is utilized as the transmission technology (advantage: higher throughput) or QPSK—Quaternary Phase Shift Keying—is utilized as the transmission technology (advantage: more robust, including Forward Error Correction). The return path utilizes a lower frequency range than the receive path and it is here where the technological concerns of the cable modem arise. The concerns are twofold: the shared cable wiring infrastructure and the traditionally one-way transmission direction. We'll take each in turn.

Shared wiring infrastructure

The concern with the fact that cable companies have a shared cable infrastructure stems, literally, from its tree and branch topology. Cable wiring, with regard to the transmission of one-way, downstream, identical signals that are used in everyday CATV viewing, is economical and appropriate for such uses. It allows for amplifiers to be placed along the cable path every once in a while to boost the signal to a necessary level in order to get the signal out to all the nodes. The problem this presents with regard to bursty data is that current cable modem technology will operate in localized "branches" under a shared transmission medium design, in which a community of *x* number of users will share the same 6Mhz channel for getting their bandwidth. Figure 5-11 illustrates the tree and branch topology, then shows the isolated view of a certain "branch" among which its nodes, or users, must share bandwidth.

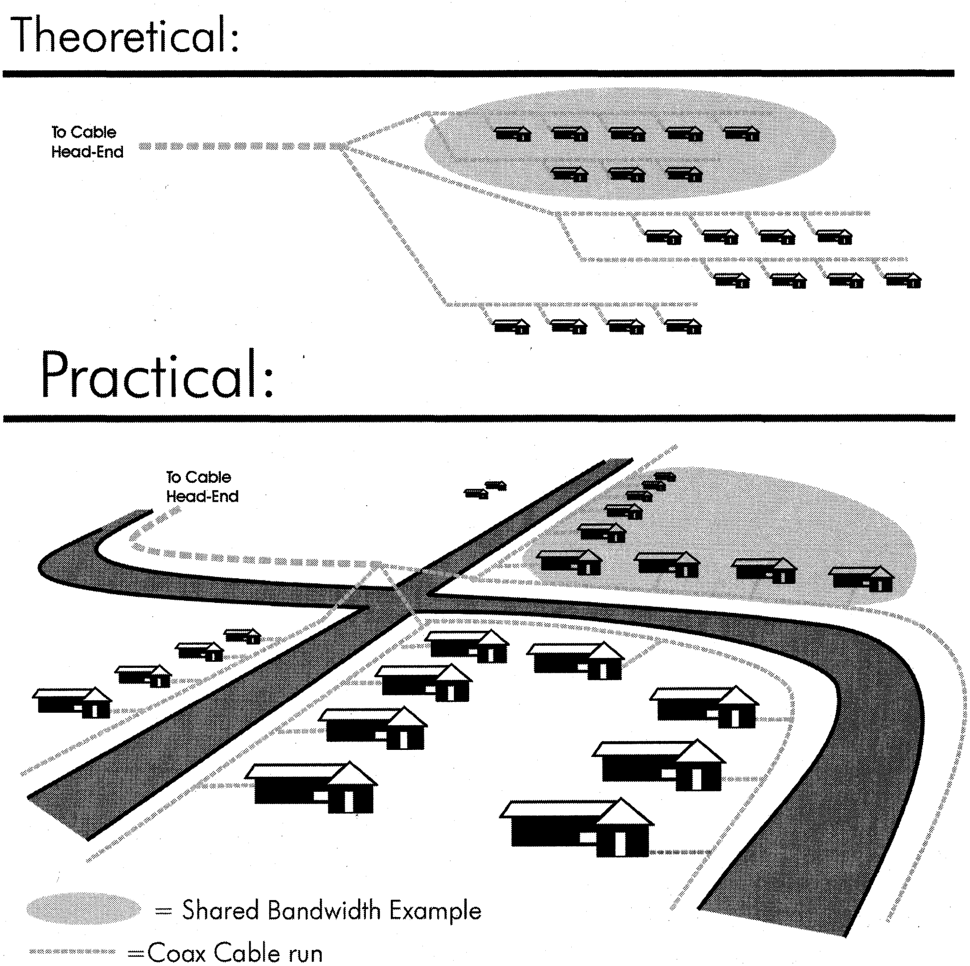


Figure 5-11: Tree and Branch Topology of CATV Wiring, and the Issue of Sharing Bandwidth Among Residences on a Certain Branch of the Cable Network.

Is this really a valid concern? To a certain bandwidth utilization, the answer is no. The means of regulating access to the shared medium (in this case, the 6Mhz channel devoted to data) employed in many CATV systems is reportedly efficient, meaning that the 10Mbps (or 36Mbps) can be utilized even with many nodes transmitting near the maximum rate. 10Mbps is a lot of bandwidth for the home, unless you're using it for lots of applications (movies, Internet access, and telecommuting) or there's a hot new killer app out that requires lots of bandwidth and all your neighbors have it. Whether or not it's a valid concern, having to share the bandwidth with the neighborhood isn't too appealing; if your throughput depends on your neighbor not using their access too much, that could make your area a bad neighborhood.

Another questionable issue regarding shared wiring has to do with how far up the trunk data must go in order to get to the head end (the place where this data is going to be redirected to wherever it's going, like the Internet). The farther up the branch you go, the higher the number of users who must share the bandwidth. At some point the requirements will be too much and it is there where some sort of transmission medium, such as fiber, must be taken to get more bandwidth closer to the neighborhood.

The last question to pose is: When has there ever been enough bandwidth, at any level of the network, for any amount of time? If you've ever had to suffer through waiting on the cable company to fix your line because you didn't want to miss an episode of *Friends*, imagine if you had to endure that same wait for your mission-critical and career-critical corporate access.

Traditional one-way transmissions

To get the obvious out of the way: Cable is traditionally a unidirectional transmission and its infrastructure has been built around that premise. Also, cable head-ends are generally islands that exist as the products of one-way transmission mediums; in other words, they aren't necessarily connected to other cable head-ends, making data exchange between and among them not immediately available and not intrinsic to their infrastructure. In contrast, every CO in the world is interconnected in one way or another, and prewired for bidirectional communications. This fact—that cable companies are traditionally downstream-centric or unidirectional—lends itself to other concerns.

If we revisit the earlier diagram that outlines the tree and branch topology of the traditional CATV wiring infrastructure, this time looking a little closer at the means by which the content signal (the TV channel signal) is propagated down the tree and to all the branches, we see that the signal is boosted along the way by amplifiers. This fact starts to dig into the wallet issue surrounding cable operators and their ability to provide the hardware upgrades necessary for Residential Broadband over cable, as shown in Figure 5-12.

Notice that these amplifiers are pointing in the downstream direction. The implications of this fact are that cable operators, in order to provide residential broadband services, are going to need to replace those amplifiers with amplifiers that can send data both ways, or augment their downstream amplifiers with upstream amplifiers (much like telecommunications COs will have to outfit themselves with ADSL modems). The difference is that the CO can outfit itself with enough ADSL modems (or ADSL line multiplexer capacity) to cover its subscribed users and add more modems/interfaces as demand merits. With cable amplifiers, proper amplifier additions must be added before even one downstream customer can subscribe to the service. Not an impossible task, just something that must be done; however, the economics of Internet users wanting more bandwidth are convincing and compelling reasons to complete such a task.

A couple of other concerns many people share with regard to cable operating companies are network management and general market perception. Unfortunately, the perception of both is not positive.

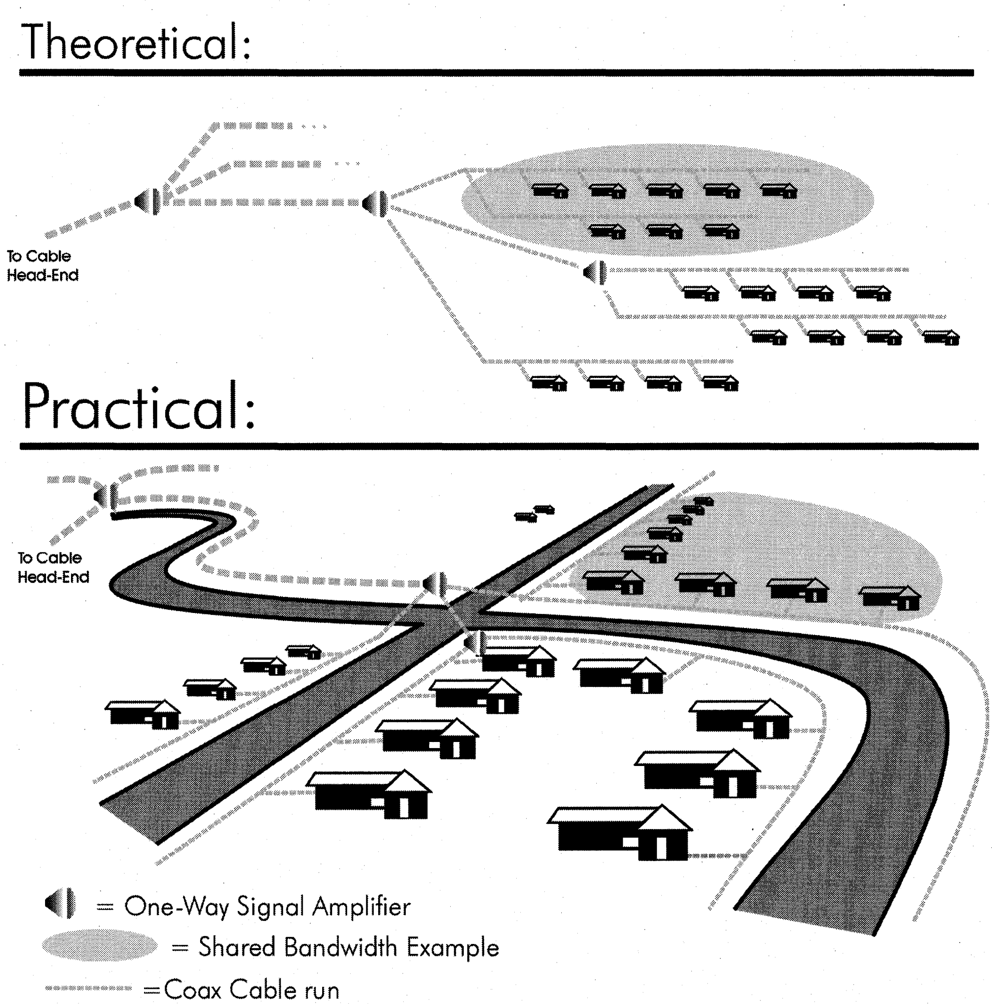


Figure 5-12: The Tree and Branch Topology, with the Amplifiers That Boost the Signal to Get It into the Neighborhoods.

One thing that cable modems do have going for them, however, is content. The cable companies are the kings of content, and once residential broadband kicks in and bandwidth enough to get movies on demand, services on demand, all sorts of other data on demand, and pay-per-view prize fight equivalents is available, there will be few who can compete with the content delivery experience cable companies have.

WAN Technologies

The following is intended to familiarize you with WAN technologies and their applications, fundamental behavior, and market implementations. It isn't a full dissertation on any of the technologies, so if you're interested in knowing facts such as which bit in the header of a Frame Relay PDU constitutes its candidacy for being dropped when the EBR for a given node is exceeded (the DE bit, for Discard Eligibility), you'll need to look elsewhere, because that isn't the intention here (it's the second bit, after the flag).

Entire books (such as the bookstore inhabitants mentioned earlier) can and are committed to the detailed treatments of each of the following WAN technologies; such detail doesn't further the mission of this chapter, which is to familiarize you with remote access technologies (including WAN technologies) to the point of being conversationally familiar with them and enable you to understand them when you're developing remote access applications. It's context-based knowledge and the imparting of such knowledge is the overall goal of the WPRS, after all.

X.25

We're starting this section with the genesis of WAN technologies, the beginnings of the WAN as a standardized means of providing wide area access for data networks. That first, old, widely deployed technology is X.25.

If you remember one thing about X.25, remember that it has intrinsic data integrity checks throughout its network "cloud," the overhead of which introduces latency and makes X.25 less desirable for transmission between and among today's powerful desktop computers. If you remember two things about X.25, then also remember that it is not a standard for a public packet-switched network; it is a recommendation for interfacing with a public packet-switched network.

X.25 was created as a result of an ITU-T (the CCITT back then) study group charged with defining a standard interface recommendation for a public data network; to the companies that needed such a service, it promised a means of avoiding the inhibiting proprietary network protocols in use at the time, provided by the likes of IBM (many different protocols from IBM), DEC, and others. It also meant a standard to which access devices for different vendors' equipment could be manufactured, against which such devices could be tested for compliance, and by which different types of equipment, made by different manufacturers, could use a common carrier to send their data across wide distances. It was also a means by which such user-requested features such as Quality of Service could be implemented (QOS is an old technology for WAN technologies, but a relative newcomer to the realm of LANs).

X.25 has been widely deployed and used over the years, both with public networks and private network implementations, because of its "abstraction" characteristics and because it generates a network cloud within which connections can be made with other devices that are connected to that cloud. The result is the creation of a common and centralized connection arena, or in more common terms, a public data network. The advantages of such a public network were two-fold and certainly economic: Rather than having to create a private network with expensive (and almost always grossly underutilized, though wholly paid for) leased lines running from each node to which connectivity was required (a mesh network), only one connection for each node was required. By creating a standard access protocol (okay, a recommendation), different types of computers, mainframes, or terminals could connect to the network and send their data; there was no need for separate networks for each type of device or each proprietary protocol. It put the means by which access was gained at arm's length, and it allowed for a pooling of network resources, which in turn resulted in lower costs.

Figure 5-13 outlines how a public data network, such as an X.25 network, can reduce the costs of access when many nodes require connectivity to many other nodes; or in simpler terms, the connectivity requirement of the network is many-to-many.

The actual recommendation from the ITU-T (CCITT) came in 1974. It was since revised in 1976, 1978, 1980, 1984, and in its "Blue Book" recommendation of 1988, which is today's most common implementation.

Notice the term recommendation instead of the term standard. The ITU-T isn't in the business of providing standards and instead provides recommendations, which the industry then promptly takes and calls a standard. Though it generally becomes a standard, the line of recommendation versus standards is clearly not crossed, for reasons such as endorsements, walking the middle-line...you get the idea.

Though X.25 has a lot of good qualities to it, among them its cost-effectiveness and wide availability, it does have limitations that are more a result of changing computing power and network architecture, as well as reduced tolerance of technology by today's compelling applications, and less a result of problems with the technology itself.

X.25 Technology Overview

X.25 is a connection-oriented WAN technology, which means that "calls" are initiated, placed, and then dismantled as a matter of course for sending data from one node to the other, similar to our telephone network and dissimilar to today's LAN technologies. In order for most PCs to interface with an X.25 network, a PAD (Packet Assembler/Disassembler) is required. X.25 utilizes 128 or 256 byte packet sizes, which are too big to be good for voice and video applications and too small to be optimum for native data network formats. With the move toward multimedia content delivery and interaction over the network (which would include the WAN link, certainly), such limiting factors—latency and less-than-ideal packet size—don't put X.25 in the very small pedestal that will hold the WAN technology of the future.

As mentioned earlier, X.25 does a number of checks on any given packet as it passes through the X.25 network, which creates a delay (compared to networks that do very little checking of packet integrity, such as an Ethernet LAN) in the overall delivery of the packet. Simply put, the checks X.25 performs take time, and that time accumulates as the packet crosses an X.25 network. This factor is perhaps one of the most limiting aspects of X.25 and will ultimately spell its demise in the face of other WAN technologies. There is a reason for this, however; when X.25 first came onto the scene, there was a need for such data integrity checks.

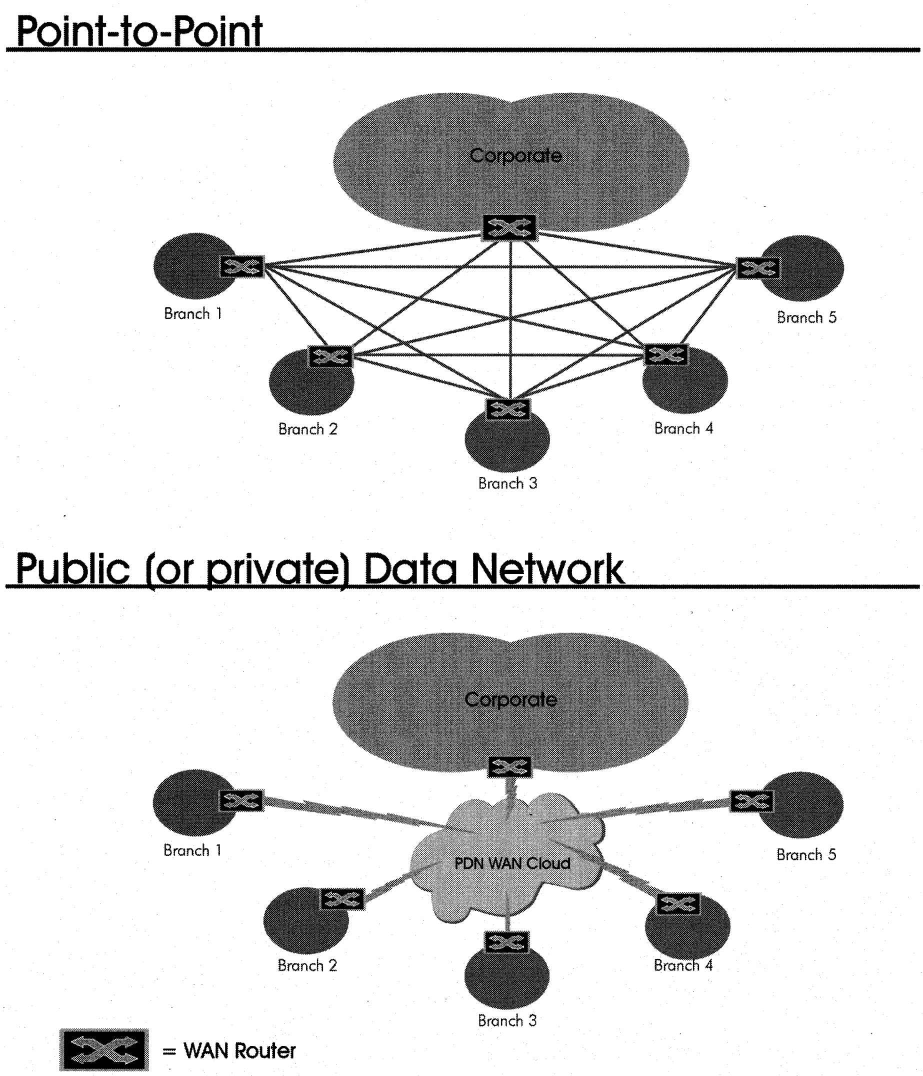


Figure 5-13: The Difference Between Using Leased Lines and a Public Data Network.

X.25 was created when the devices utilizing its services were, compared to today's standards, processor-poor. The requirements of the data network, in the time it was created and even revised, included a need to ensure the integrity of the data that crossed through its network cloud. That meant that checkpoints for data integrity at each stop (hop) along the network way had to be a part of the network, and when integrity checks are done at the packet level, the requisite overhead and consequent latency is significant. Today's computers, with their 300 million or so cycles per second, don't require such hand-holding transfers, because they have the computing horsepower to implement data integrity (error) checks and balances upon receipt of packets. If errors occur, the receiving computer simply lets the sending computer know of such errors (through NAKs) and requests the appropriate response, such as a retransmission.

T-Carrier

T-Carrier facilities have been around for over 30 years, and were designed as a means of digitizing and transmitting multiple voice channels over twisted pair media (multiplexing), increasing overall telephone network transmission capacity.

A T1 line, by definition, is a digital transmission facility that provides 24 digitized channels over two twisted pairs (a total of four wires). As time has gone on and T-Carrier services have been widely used, the differentiation between the widely used T-Carrier facility and its throughput levels—transmission capability levels more accurately described with DS0s or DS1s—began to muddy. Today, many people intermingle the term T1 among carrier type and throughput capability, which tends to confuse the understanding of the technology.

Thus, T-Carrier is a tricky bit of work, since its name is used to denote transmission signaling, throughput rates, and the carrier system itself. So if you say, "I have T1 access to the Internet," that could be interpreted to mean you have a dedicated T1 line that connects you to an ISP, and the equipment on either end creates one big 1.536Mbps pipe, or you have a Frame Relay connection to the Internet that runs over a T1 line, which operates at 1.536Mbps. Which do you mean? Either would be correct, though technically it would be more accurate (and descriptive) if the latter were to say, "I have a Frame Relay connection to the Internet. It's running over a T1." Does anyone wonder why T-Carrier technology can get a bit confusing?

So to reiterate and conclude in one sentence: T-Carrier both defines the transmission medium (over copper) and is a defined transmission signaling technology (24 DS0 channels of 64kbps each).

T-Carrier Technology Overview

The T-Carrier facility is based on DS0 (the 64kbps digital payload), which in this discussion will be called a channel. Note that the DS0 is a digital transmission facility.

A T1 is divided among 24 individual channels, each of which is generally used to support one telephone conversation, one analog modem connection to wherever, or one fax transmission coming in from your favorite office supply shop—the point being that a T1 provides 24 virtual "telephone lines," and what you do with them depends on how that T1 is used. Through the use of compression, some applications of T1s can squeeze more voice channels out of one T1 line, but that application is in the PBX and voice end of things, not the remote access end of things. With remote access applications, you'll be getting 24 digitized 64kbps channels out of your T1. Because T1s are so prevalent, the subject of T1s and how they work merits some more depth; see the section titled *T1s, E1s, PRIs and All Those Bits*.

T-Carrier facilities use a signaling mechanism called Time Division Multiplexing, which is best explained by an analogy. Imagine you have a line of trucks taking payload to a destination in preordained, very specific intervals (precisely 8,000 times per second in this case, which is a pretty fast truck), and the trailer (open at the top in this example) has exactly 24 evenly spaced slots into which cargo can be placed. Now let's say that you have 24 different companies that have reserved a spot—the same spot on each truck—within those 24 slots for the transport of their payload. When a truck starts moving out, it is loaded with its payload and heads across the T1 highway toward its destination. Thus, the 7th spot on the first truck's trailer is occupied by payload from Company XYZ, as is the 7th spot on the second truck, the third truck, and so on, until Company XYZ hangs up its cargo contract and no longer wants to transmit its goods across the T1 highway. The 7th spot on all the trucks then becomes available. We could call this the Channelized Trucking Company, since its trailers are all divided among 24 individual channels.

A T1 behaves in a similar manner. Each of the 24 channels on a T1 has a specific payload capacity, which is equivalent to a DS0 (or vice versa, meaning that a DS0 is equivalent to the payload available on one channel of a T1—it's kind of a chicken and egg deal), or 64,000 bits per second.

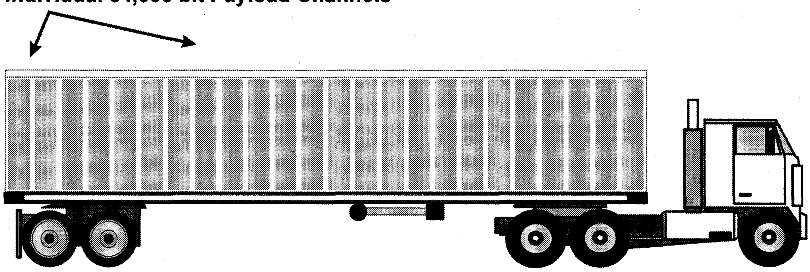
This is all great and interesting if you're putting together a rack of modems that need individual lines, but we're in the WAN section, and we want one big pipe to the Internet, a specific remote location, or wherever; we don't want it divided. How does that work? Well, let's take a look at the Unchannelized Trucking Company to find out.

Back to the line of trucks waiting to take payload to their destination across the T1 highway. With the Channelized Trucking Company, each of its trucks had trailers that were divided into 24 separate slots, into which a company such as Company XYZ could place its payload of up to 64,000 bits per second. In contrast, the Unchannelized Trucking Company has trucks with trailers that aren't divided into separate slots, and instead have the full 1.536Mbps of available payload to make available to one customer (for the reason why a T1 has a payload of 1.536Mbps instead of the generally stated 1.544Mbps, see *Technical Talk: T1s, E1s, PRIs and All Those Bits*). Figure 5-14 shows the difference between these trucking companies.

As mentioned earlier, the T-Carrier facility is based on its transmission of the DS0 (the 64kbps digital payload) in increments of 24, which when aggregated into the T1 frame format becomes the basis of the entire North American Digital Hierarchy in the form of the DS1 (1.544Mbps). Figure 5-15 outlines the DS hierarchy and their corresponding voice channel capabilities.

Channelized Trucking Company

Individual 64,000 bit Payload Channels



Unchannelized Trucking Company

One big 1.536Mbps Payload Channel

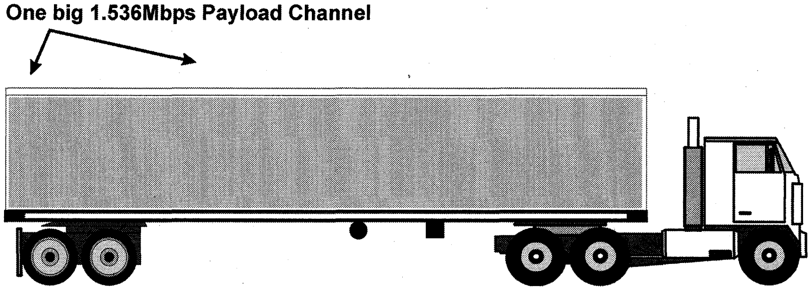


Figure 5-14: The Channelized Trucking Company and Its Payload Division Versus the Unchannelized Trucking Company and Its One Big Payload.

North American Digital Hierarchy

Digital Signal	Throughput (Mbps)*	Channels (DS-0s)	Equivalent T1s
DS-0	.064 (64,000 bps)	1	1/24th
DS-1 (T1)**	1.544	24	1
DS-1C (T1C)	3.152	48	2
DS-2 (T2)	6.312	96	4
DS-3 (T3)**	44.736	672	28
DS-4 (T4)	274.176	4,032	168

* Throughput rating includes administrative signaling.

Figure 5-15: The North American Digital Hierarchy and Its Corresponding Throughput Capabilities and Voice Channels.

T2s are uncommon except in movie sequels. Generally, T1s are used until a throughput requirement somewhere around a T3 is required.

T1s, E1s, PRIs and All Those Bits

You've heard talk of T1 and E1, and you may have heard that they don't have the same bandwidth capabilities and are not compatible, but you may be wondering: What's the difference, and why have two similar kinds, if not for entertainment/confusion value? Good questions, and a good subject for a Technical Talk.

First we'll define the T1 in technical terms: A T1 consists of 24 DS0 channels. Each DS0 carries 64,000 bits of information per second, and with the addition of one control bit per T1 frame (the 193rd bit of a T1 frame), we get a total transmission rate of 1.544Mbps (1.536Mbps of which is available to the user).

In the North American digital signal hierarchy, a DS1 (Digital Signal level 1) is equivalent to 24 DS0s (a T1), and the telecommunications infrastructure in North America is based on that hierarchy. (See Figure 5-15) Different parts of the world, however, have developed their own, different digital signal hierarchies.

In Europe, the European Hierarchy defines a DS1 as carrying 30 DS0s. In Japan, the Japanese Hierarchy defines a DS1 as 24 DS0s, but defines a DS3 as 480 DS0s, versus the North American Hierarchy which defines a DS3 (T3) as 672 DS0s. Thus when there are discussions about T1s and E1s, and their differences in bandwidth and voice/data channel handling capabilities, the reasons for their incompatibilities and the need for different interfaces for each become clear.

Back in North America, where T1s live in close quarters with ISDN PRI interfaces, the differences between the two require a closer look. The difference between the transmission characteristics of T1s and PRIs are that (as we know) T1s utilize 24 DS0s and add a bit to the T1 frame for control, whereas PRIs utilize exactly 24 DS0s, but reserve the last (24th) DS0 for the ISDN D Channel use. Figure 5-16 illustrates the difference.

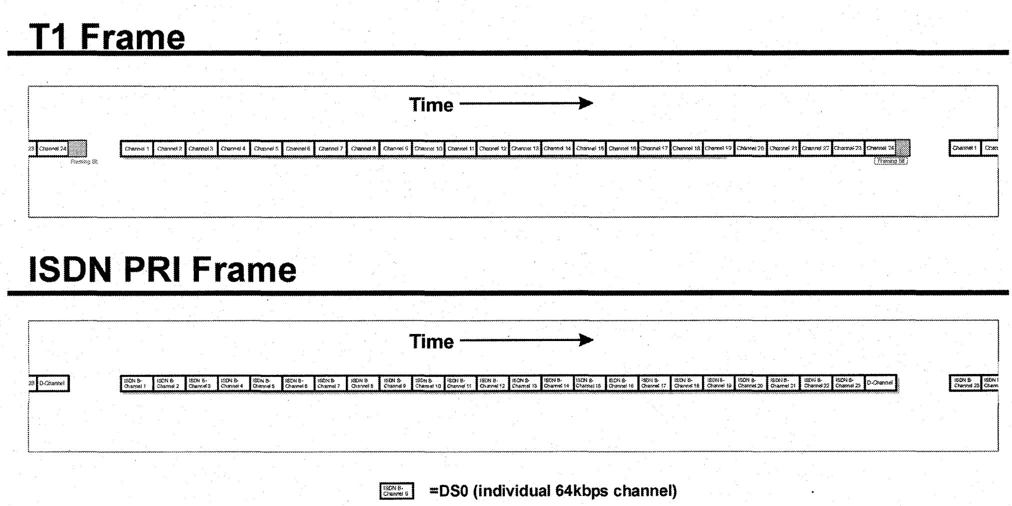


Figure 5-16: The Difference Between T1 Frames and PRI Frames.

Because the ISDN PRI utilizes one of the DS0s for its signaling, it doesn't require the additional control bit to be added to each frame as T1s do. Thus, you may see bit (not big) differences between transmission rates of T1s (1.544Mbps) and ISDN PRIs (1.536Mbps), despite the fact that they both "utilize" 24 DS0s.

ISDN PRI Technology

ISDN PRI technology was discussed earlier in this chapter. We won't beat it up again here in too much detail, though we will go over some (WAN-centric) concepts not covered earlier.

As mentioned previously, ISDN has similar bandwidth capabilities as a T1, and uses the same 64kbit DS0 as its basic building block. ISDN provides comprehensive administrative capabilities much better than a T1. The technical aspects of how it implements its administrative services are too involved (and provides definitions, not explanations). It can be loosely introduced by stating that ISDN has its own management "language," which ISDN equipment (more accurately, ISDN Network Terminators, which are built into ISDN equipment) understands and can respond to, providing a native and inherent management structure within the technology itself. This is sometimes referred to, in ISDN and other technologies, as a management layer or a management "plane." For those who want more technical information, and even more terminology, this management comes in the form of Q.931 messages and is carried within the LAPD frame.

Like (one aspect of) T-Carrier, ISDN PRI is a signaling standard. Though ISDN isn't the WAN technology or common carrier of the future, it has immediate, real-world applications today and is being implemented, more so recently, as increases in ISDN popularity have been making incremental appearances in the residence and office.

Despite ISDN's seemingly sputtered break into the WAN and the home, what can be attributed to ISDN's success and track record is the use of control messaging (Q.931) and call setup/teardown (LAPD) for the WAN; both have been implemented in one form or another with stronger WAN candidates such as Frame Relay and ATM.

For technical details on how ISDN PRI differs from T1s and even E1s, see the previous section titled *T1s, E1s, PRIs and All Those Bits.*

ISDN's competition and implementations as a WAN technology are quite similar to T-Carrier. Though its future isn't doomed by inadequate administrative facilities, other more attractive WAN technologies have more going for them than ISDN, and in comparison to the two following technologies, ISDN's future is not destined for big things, but it's certain to be around for some time to come.

Frame Relay

Frame Relay could be called the modern makeover of X.25. Frame Relay came on to the WAN technology scene in the early 1990s, though standardization groups began work on it in the late 1980s, and has been growing in its installed base since its introduction to the market. There are a lot of things about Frame Relay that make sense, and it solves most of the problems other WAN technologies run into, including bursty traffic handling, administrative facilities, QOS capabilities, upper throughput range, latency (resulting from network "touching" of packets), wasted leased-line bandwidth and costs, and others. Is it the golden WAN technology? Some people would give you a very quick and resounding NO to that question, many would say it probably isn't. I think it's hard to say at this point, but Frame Relay has a lot of attractive characteristics, is placed in the overall telecommunications network scheme in such a way that allows the BISDN infrastructure to augment it, not replace it, and has throughput limits that seem to provide a lot of headroom.

Frame Relay Technology Overview

Like X.25, Frame Relay is a connection-based recommendation. A Frame Relay network is a public (or private, or some combination of the two) packet switching network, whose most appealing characteristic is that it does very little, in fact as little as possible, to the data that gets sent across its network. Instead, Frame Relay relies on end nodes to provide error correction, ACKs, NAKs, sequencing, and other processing-intensive operations. Frame Relay was designed to do as little as possible, and in so doing, keep latency across its networks to a minimum and the cost of its core network hardware to a minimum (cost savings which are passed on to the user).

Similar to X.25, Frame Relay is an interface standard, and says nothing about the internal workings of the Frame Relay cloud. For end users, that's fine: We don't care what happens in the core network cloud, as long as we can get and send our data quickly, efficiently, cost effectively, and with lots of throughput.

The most attractive aspects of Frame Relay include the following:

- Frame Relay does very little to the data that passes through its network, which results in much lower latencies than X.25, though not as low as ATM.
- Frame Relay is based on a mesh network instead of a point-to-point network, which makes connection to a Frame Relay network much more economical than leased-line alternatives.
- Since Frame Relay networks do less processing to their data, implementation of Frame Relay networks are more cost-effective than other, more processor-intensive WAN solutions.
- Frame Relay has the ability to move data at T3 (approximately 45Mbps) rates and even slightly higher rates.
- The maximum Frame Relay PDU is 4,096 bytes and is variable, allowing LAN frames to get Frame Relay headers prepended and then sent on their way (no slicing and dicing of the original frame).

Frame Relay uses an economical approach to the transmission of data called Statistical Time Division Multiplexing (STDM). STDM is similar to the Channelized Truck Company discussed in the T-Carrier section (which used TDM), with a few important distinctions: The Frame Relay Company's trucks don't leave at preordained intervals and will carry any payload in their slots. Also, their trucks' payload slots are not necessarily constrained to specific sizes. There is also a buffer near the loading dock that can store payload for a certain, small amount of time. If it helps (since Frame Relay often runs over T-Carrier), you can consider Frame Relay over T-Carrier in the following way: When Frame Relay is in charge, the Frame Relay Company takes over management of the loading dock and cargo-placing booms, and is more flexible with its payload and scheduling requirements than the Channelized Truck Company. Though they may use the Channelized Truck Company's trucks and trailers, they allow their customers' varying-sized payloads to be dropped off with them, and then they (the Frame Relay Company) deal with stuffing those varying-sized payloads into the compartmentalized trucks of the Channelized Truck Company, and also deal with unloading (and putting back together) when the trucks reach their destination.

Frame Relay works on the basis of Committed Information Rates, Committed Burst Rates and Excess Burst Rates (CIR, CBR and EBR respectively). That means that bursty networks such as LANs can get Frame Relay service at a certain CIR and exceed that rate during bursty periods up to the CBR or EBR (extended periods at or above EBR will make your data eligible for being dropped) without having to waste financial resources on leased lines that equate to the EBR. For example, you might have a 512kbps Frame Relay CIR that's been brought to your premises via T1; if your corporation at times exceeds 512kbps, perhaps up to 1Mbps under certain conditions, the Frame Relay interface will handle that excessive data. If your EBR were 1Mbs and you had bursts that were hitting 1.2Mbps, then 1.2Mbps would be eligible for being dropped within the Frame Relay network. The advantage of Frame Relay's ability to handle bursty traffic, in this situation, is that you don't have to lease an expensive T1 line to get burst rates of 1Mbps or 1.2Mbps; if your sustained average throughput is 512kbps, you can base your usage on that rate, not on your peak, or burst, rate. There are other, more sophisticated means of provisioning peak rates in Frame Relay, which have to do with buckets and credits, but the details of such algorithms are outside the scope of this discussion.

ATM

ATM stands for Asynchronous Transfer Mode, and has been positioned as the underlying technology to take networking—both data networking, video transmission, and telecommunications—through the 21st century, all on the same wire. And if the amount of planning, theorizing, debating, refining, and general thought that has gone into ATM is any representation of its chances of achieving that lofty intention, then ATM's chances are good.

ATM, however, can be intimidating, often because of the sheer volume of dry reading or research that must be done to achieve even a reasonable familiarity. The result, too often, is a break after only sipping on its details, from which many never return.

This treatment of ATM is an intentional departure: It's been structured to explain *why* ATM is the way it is, and by doing so should let you get through it with the least amount of pain or sleepiness.

Getting to ATM

We've gone through technical overviews of other prominent WAN technologies already, and we've seen a sort of trend. X.25 brought the cost-effectiveness of standardization and shared mesh topologies to data networks; T-Carrier utilized a digital telecommunications infrastructure, and the well understood T-Carrier technology, to get data moved from point to point using the existing and ubiquitous PSTN. Frame Relay improved on both, taking the attractive shared network packet-switching attributes of X.25 and the low-latency attributes of T-Carrier, and then threw in its own added features to make it a great solution as a shared mesh data network for today's high-speed client computing. And in the beginning, the middle, and still today, there was the need to transmit plain old voice data throughout the world.

We also found that there is another network sending out data of one sort or the other, which is the CATV network. It utilizes its own means of moving data, whether that's movies, digital music, or 24 hours of television shopping, which implements none of the transmission technologies discussed above.

But there has been something missing throughout all of this; a fundamental cohesiveness that all these WAN solutions and voice transmission facilities lack. What's missing is a common network, certainly, but also a common carrier, which is the aim of ATM. ATM strives to be a common carrier for voice, data, audio, video, and any other data that can be transmitted over one network that would become the Information Superhighway. You name it, and ATM wants to be able to send it, and has been designed to be able to do just that.

Creating the Common Carrier's Shopping List

In order to be the common carrier of data, voice, video, and any other type of data, ATM must provide all of the services each of the data types need, but must do so within the constraints of one data type.

The difficulty with trying to please all of the people (or data types) all of the time centers on the fact that different data types are best serviced at different sized PDUs. Voice is best served by small packet sizes, such as 32 bytes per PDU, while "computer" data is served best my much larger sized PDUs (Frame Relay, a data-centric technology, has a maximum PDU of 4,096 bytes). Thus, there is a disparity between voice and data. How do you solve these differences? First, you must be very fast; so fast that the compromised (smaller or larger than you would like) size of the PDU is grossly outweighed by the increase in speed or throughput. Second, you must promise and deliver compelling reasons—real world reasons—why changing from the status quo is worth it in the short term, the near future, and the long run.

The means of dealing with the difference in optimum PDU size is: being fast and being everywhere. That brings us back to the "one data type for all" philosophy. Why one data type? Because one data type, with a fixed length and fixed header sizes, would enable that same data type (regardless of its content) to traverse the network quickly, efficiently, and in hordes and hordes, gigabits and terabits at a time. It is so efficient to use fixed sized PDUs that the switches that forward them can actually function and process at rates higher than the line speeds themselves. That's fast switching. Even if you have to chop up larger PDUs from their native format (like Ethernet with its 1518 byte maximum PDU) into smaller PDUs to utilize the network, the benefits of the anticipated cost effective and higher bandwidth WAN service availability and low latency associated with the smaller PDU implementation make the work involved in chopping up the data (and reassembling at the other end, if necessary) worth the effort.

The means of providing compelling reasons for changing from the status quo are somewhat less immediately tangible, but certainly are at least as important as all the technical reasons combined. In the short term, the common carrier can actually concentrate on a subset of its strengths: the ability to move data in large volumes. One short term use might be upgrading existing LAN backbones to the higher capacity capabilities of a common carrier technology. A mid-range or near-future reason for utilizing the benefits of a common carrier might be to augment the "coming of age" of multimedia applications to the desktop. This movement requires a significant amount of bandwidth, and also a means of guaranteeing a certain level of service (voice and video over data networks exist today, with Internet phones and monitor-top desktop cams, but they're jittery and hog all of the available bandwidth, and generally speaking, at best are novelties rather than real solutions). For the long term, compelling reasons for a common carrier include all of the preceding reasons, as well as the ability to turn multiple information service networks into one cohesive delivery platform. This is one compelling reason to move to a common carrier; if you could take advantage of using one network even for data and voice (and thus make it more cost effective in terms of service charges, administration, application and content development, and new market potentials), concessions would be made to integrate those services.

When we put these requirements of a common carrier into shopping-list form, the end result, much more concisely presented, looks something like the following:

- Carry all sorts of different data, including voice, "computer" data, video, and others.
- When carrying that data, allow users to request various levels of "service" so that information delivery that is sensitive to delay, bandwidth constraints, or timely sequential arrival can be accommodated.
- Carry the data in large volumes, quickly, and efficiently. In specific terms, provide for lots of bandwidth, low latency, and make sure switching infrastructure processing power ("inside-the-network-cloud" equipment efficiency) isn't prohibitively expensive.
- Be media-independent, allowing existing transmission facilities (copper, fiber, or co-ax) to migrate without making expensive physical changes to their infrastructure.
- Create the ability to merge all the various information networks, including voice, data, video, into one network.

- In merging those networks, allow graceful handling of different transmission characteristics, such as bursty transmissions (variable bit rate) versus continuous transmissions (constant bit rate).
- Allow for an incremental migration from other technologies; avoid the requirement of an "all or nothing" approach.
- Be designed in such a way that limits the likelihood of being outdated in the near future. Don't be an interim solution, be the long term solution.

We can boil these requirements down even further if we try to:

- Carry all information data types efficiently and meet the different transport requirements of each.
- Be available over any transport media and provide mechanisms to interact with existing technologies.
- Be the transmission technology for the long term future of information delivery.

One issue that was alluded to but not directly addressed comes last. We have worldwide PSTN interconnectivity, which means you can call someone across the globe just as easily as you can call your neighbor across the street. Being the common carrier of the future of information delivery necessitates that information boundaries, in a world where the economy is global rather than local, be non-existent. It further requires that such technology not be implemented in one way in North America, another way in Europe, and another way in Japan and Asia. This brings us to the last item on the common carrier shopping list:

- The common carrier must be a worldwide standard.

ATM Technology Overview

With all those shopping list items, you can imagine the difficulty coming up with a technology that met all the requirements. Perhaps an even more challenging task would be choosing among the different ideas and methods, often heatedly defended and promoted, for going about achieving such a standard. An international body comprised of industry leaders in both the telecommunications and data industries, those who (choose the word as you will) created, devised, invented, or standardized ATM, have done it. The means by which ATM reaches those lofty goals is the subject of the following sections.

Carrying all data types

The means by which ATM carries all data types in an efficient, fast-switching, low-latency means is by having a standard sized ATM PDU, called a cell, of 53 bytes. Hereafter, the ATM PDU will be referred to as a cell, much like an Ethernet frame is often referred to as a packet.

The ATM Cell. An ATM cell is always 53 bytes. An ATM cell always has a 5-byte header, leaving a 48-byte payload. Always. This presents a deterministic, or specific and predictable, means of determining the beginning and end of an ATM cell, which in turn makes equipment that must handle ATM cells efficiently and quickly. The handling of all data types, including voice, data, and video, is thus done within the 53-byte cell. ATM transmission characteristics (such as service requirements, routing information, source and destination addresses, path identifiers, and payload type identifiers) are carried in the 5 byte ATM header. The information (the actual "data" that's being transmitted), plus that data's information necessary for its adaptation to ATM, is handled in the 48-byte payload. That is the crux of ATM. All other features, services, capabilities, and characteristics must do their work within those confines. Figure 5-17 puts this into a picture.

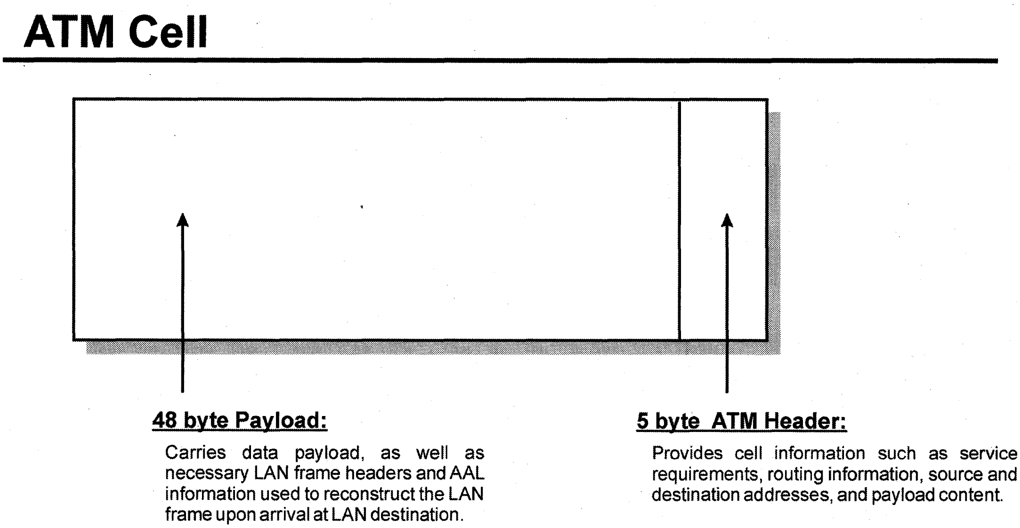


Figure 5-17: The ATM Cell, with the Division of Carried Information Between the Header and the User-Available Payload.

ATM Connections. Pick up the telephone, dial your friend's number, and you've completed a call. You have a number identifier (the telephone number), and within the telecommunications network you have a circuit assigned to your call, which is sent over larger, multi-circuit transmission facilities (multiplexed). Though ATM connections don't fit exactly into that example, they're similar. ATM connectivity works on the basis of two identification elements: the Virtual Channel Identifier (VCI) and the Virtual Path Identifier (VPI). Combined, these two channel elements constitute the Virtual Circuit Identifier.

Perhaps a better comparison is a TCP/IP address: In the TCP/IP network address 210.21.98.3 with a subnet mask of 255.255.255.0, you have the network address (210.21.98) and the local address (3) which together constitutes the IP address. Network address + local address = IP address. In ATM terms, VCI + VPI = Virtual Circuit Identifier. Both the IP address (in IP networks) and the Virtual Circuit Identifier (in ATM networks) are used for routing their respective PDUs across their networks.

The reason the telephone example is pertinent, though, is the virtue of its connection-oriented sequence. With the telephone call, the circuit is created when necessary (when the person picks up the telephone and dials the number) and torn down when they hang up. ATM works in a similar manner, though its general usage provision differs slightly. Switched Virtual Channels, or SVCs, are similar to telephone calls in that they are created when the user requests use of the network (in computer terms, your "request" might be an attempt to connect to a server on the other side of an ATM WAN link, at which time the connection would likely be initiated and made so quickly that it appears as though the connection were always "up"). In contrast to the need to initiate a call to establish the connection, Permanent Virtual Circuits, or PVCs, are always up. Examples of an SVC and a PVC, respectively, would be a dial-up connection to the Internet and a dedicated connection; with a dial-up connection you must tell your modem to dial your ISP, at which time a connection is made. An example of a dedicated connection to the Internet would be an ISDN connection that's on 24 hours a day, always connected and ready for transferring data to or from the Internet, whether any data is passing back and forth or not.

Adaptation to ATM. To be the carrier of all data types, all data types must be convertible into ATM cells. This conversion, or adaptation, is done through the ATM Adaptation Layer (AAL). With this abstraction of ATM, or in less technical terms, by outsourcing the means by which other technologies (or data types) become compatible with ATM, the technology makes itself available to any type of data. Figure 5-18 illustrates this.

Because some data-type technologies have additional information necessary to provide adaptation to ATM, part of the 48-byte payload may be dedicated to the adaptation of a given data type. Such data types have been specified and standardized within ATM technology. Figure 5-19 illustrates those AAL types.

Although the ATM cell facilitates one size for all sorts of different data types, and the AAL allows those different data types to be adapted to ATM cells, neither inherently solves the issues surrounding different data types' dissimilar service requirements. LAN data is traditionally bursty; voice traffic is traditionally a constant state. Video is sensitive to timing requirements. ATM addresses these issues through traffic classification.

Classification of ATM Traffic. The classes of traffic within ATM have been categorized, recategorized, uncategorized, and then rethought and represented. Throughout all the changes of classification naming and conventions the fundamental requirements have remained the same. Those requirements deal with which service parameters the traffic being adapted by the AAL is most sensitive to. They fall into a few categories:

- Constant bit rate requirements.
- Variable bit rate requirements, which are sensitive to timing constraints.
- Connection-oriented variable bit rate requirements, such as bursty computer-data applications.
- Variable bit rate requirements, such as bursty computer-data and Frame Relay WAN applications.

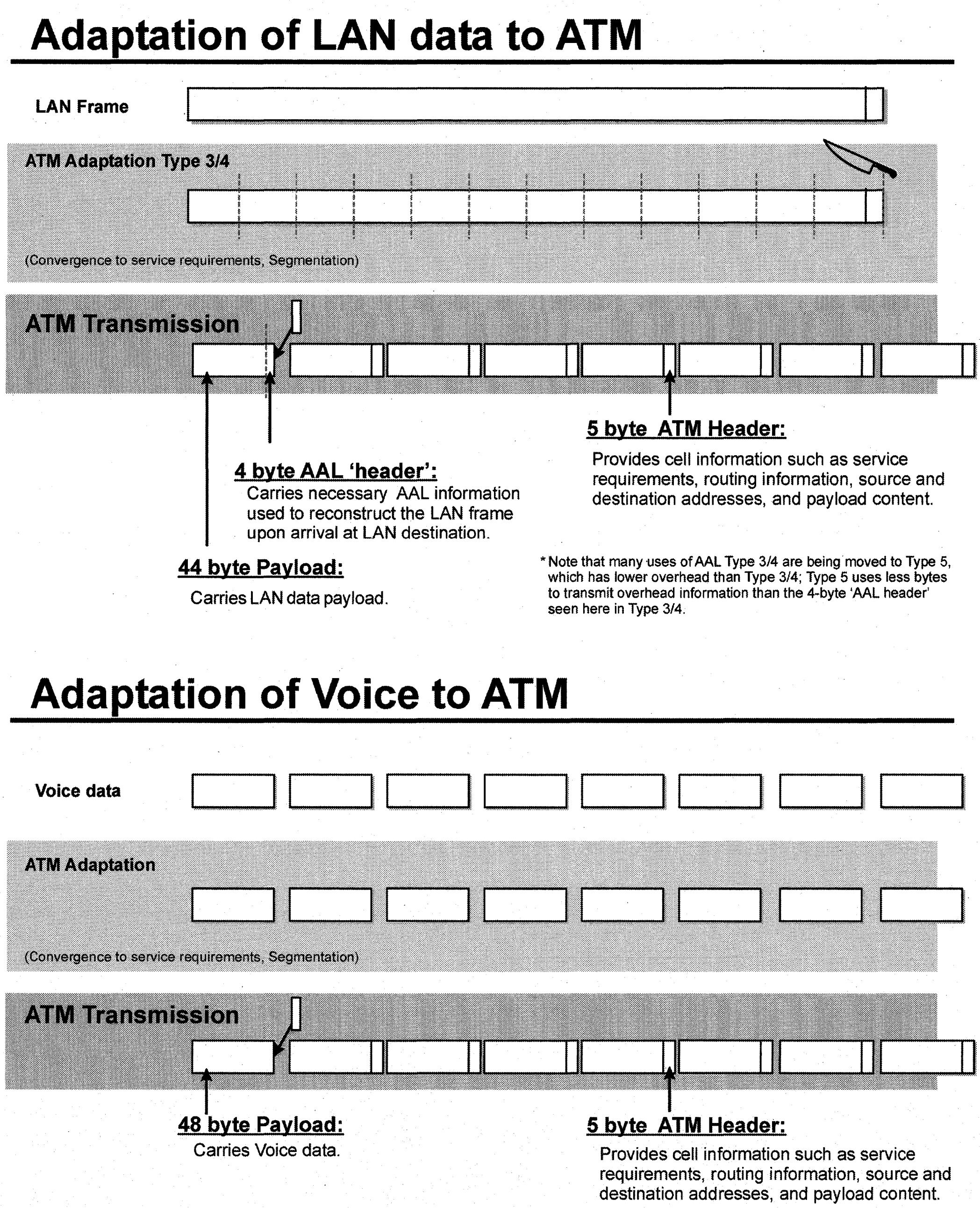


Figure 5-18: The ATM Adaptation Layer.

ATM Adaptation Layer Types

Type	Usage Summary	Data Bytes (of 48-byte ATM payload)
AAL Type 1	Constant bit-rate (CBR), time-dependent applications	46-47
AAL Type 2	Intended for variable bit-rate (VBR) applications	N/A
AAL Type 3/4*	Variable Bit Rate Connection & Connectionless oriented traffic that is tolerant to delay (e.g., LAN data). Intended for data requiring some sequencing or error detection.	44
AAL Type 5*	Variable Bit rate connection oriented data requiring minimal sequencing or error detection (contrast to Type 3/4). Used in support of upper protocol (such as Frame Relay) transmission over ATM.	Variable, but more than Type 3/4 (44)

* Type 3/4 and Type 5 currently garner the most interest. Types 1 and 2 were initial (and not widely used) definitions.

Figure 5-19: ATM Adaptation Layer Types.

These classifications were grouped into Types, such that there were Type 1, Type 2, Type 3/4 (Types 3 and 4 were combined), and Type 5 standards established for adapting different data types to ATM. Each different type has a specific means by which data is placed into an ATM cell's payload. For a technical example, a Type 1 PDU starts with a 4-bit sequence number (placed at the beginning of the payload part of the cell), then has a second 4-bit sequence dedicated to providing error correction to the first 4-bit sequence, then has an optional 8-bit (one byte) pointer field (its use or nonuse identified within the initial 4-bit sequence field), leaving the 46 or 47 bytes available for actual data. Why the technical example? The importance of the example is that a Type 5 does not have the same 4-bit, 4-bit, then optional 8-bit fields in its payload field; these types specify how data is segment and "formatted" into an ATM cell, in order to best accommodate different data types individual service needs.

Thus, while ATM has a standard 5-byte header and 48-byte payload, the format of the payload differs among different AAL "Types." Yes, my nose is getting a little fizzy too. But such classification of types for ATM allows switches to make very quick decisions regarding the servicing of a cell based on its type, which contributes to and facilitates the overall ability of ATM to provide the appropriate quality of service to many different types of data, all within the same transport technology, or to keep with our terminology, allows ATM to become the common carrier.

Whew. So all these attributes of ATM—the size of the cell, its means of connection to other ATM equipment, the adaptation of different data types to ATM formats, and the classification of ATM traffic—all contribute to (or facilitate, depending on your perspective) ATM's ability to carry all data types.

Media independence

Media independence is achieved with ATM because its standard does not require that a certain medium be employed. It is media independent, much like you can buy a Windows NT machine and put an Ethernet, Token Ring, or FDDI card in it and still achieve network capabilities. Thus a manufacturer, if it so chooses, could implement ATM over standard CAT 5 UTP networking cable (found all over the place in Ethernet LANs today). Or a manufacturer could implement ATM over multimode fiber, utilizing the high transmission rate traits of fiber and the fast switching capabilities of ATM to create a backbone that speeds all sorts of data over a backbone. Or a manufacturer could implement ATM over microwave transmission facilities.

Of course, there is engineering to be done to figure out how to get, say, CAT 5 UTP to transmit the electrical signals that will carry the signal on which ATM will be transmitted. With Windows NT, you cannot simply touch a network wire to the outside of the box and expect to get connectivity; you must have an interface card, engineered and designed for a certain medium such as Ethernet, installed and configured to run with Windows NT in order for network connectivity to be achieved. ATM's requirements, whether implementing it on Windows NT, a Cisco router, or in a Northern Telecom telecommunications switch, are similar.

To ensure interoperability among different vendors' products, there are guidelines created to provide an understood playing field for different media implementations. Standards exist today for the transmission of ATM over certain media including CAT 5 UTP, T1, E1, T3, E3 and fiber, to name many.

The long-term carrier

ATM's design to send all known data types is augmented with its abstracted transmission model, which allows the unknown data types of the future to fit into its model, or to be fit into its model, without having to rewrite the technology's infrastructure. This, of course, is due to the planning and future-minded engineering that went into ATM.

Another requirement for the common carrier of the future centers around its need to be relatively light in terms of processing requirements, which is a reflection of the evolution of the client computer. As mentioned in the X.25 and Frame Relay technology sections, the computer that sits on today's desktop is far and away more powerful than those that sat on desktops 10 or 20 years ago. In fact, it's more powerful than the mainframes that were servicing the terminals that were sitting on desktops 10 or 20 years ago. As a result, the processing burden for ensuring the integrity of data transmissions can be placed on the end unit, not the core network switch. For the "computer" data part of ATM's long-term common carrier candidacy, that means that ATM's hands-off approach to sending data (ATM doesn't do significant, and therefore processor- and latency-intensive, error checking on its cells as they move through the network) situates it well for long-term viability. There are other data types to be concerned with, though, such as voice.

Another advantage of the continuous leapfrogging in processor power and technology is that its benefits, in the ability to move tons of data, are also reaped in switches. That means that any data being sent over an ATM switch (not just "computer" data) is benefiting from process speed improvements. More than the quality of service available for voice, this fact becomes pertinent because of ATM's ability to handle lots of data without introducing latencies, which any common carrier that will serve information transmissions for the long term must certainly be able to do. Handle lots of data and handle it quickly, which brings us to video.

Video is inherently bandwidth hungry. It also has many QOS requirements, such as a constant bit transmission rate that the network can guarantee for the duration of the connection, which puts additional processor (and logic) burdens on the network, while ensuring very little loss. In order for a common carrier to be viable for video and multimedia transmission applications, it must have the ability to provide a guaranteed QOS so that video or audio is smooth and constant, not jittery and intermittent due to dropped or delayed cells. With the combination of video and data networks, the issue of QOS is equally important, indeed perhaps more important, for mission-critical applications as well. When bandwidth-hungry multimedia applications are utilizing perhaps disproportionate network resources, it is vital that mission-critical applications not be starved of bandwidth, latency, or other QOS requirements. ATM has the mechanisms built into it to facilitate all of those requirements.

Another requirement of the long-term common carrier technology that will prevail is that it must have administrative facilities. The need to be able to get administrative information from the common carrier of the future is a must, and ATM is well situated in that category as well.

CHAPTER 6

RAS Programming Guide

Remote Access Service (RAS) provides remote access capabilities to client applications on computers using Microsoft® Windows® operating systems. RAS client applications can perform the following tasks:

- Display any of the RAS common dialog boxes. This includes the main **Dial-Up Networking** dialog box, the **Dial-Up Networking Monitor** property sheet, and other dialog boxes for creating, editing, copying, or dialing a phone-book entry.
- Start and end a RAS connection operation using the common dialog boxes or the low-level dialing functions.
- Create, edit, or copy phone-book entries using the common dialog boxes or the low-level phone-book functions.
- Work with entries in the RAS AutoDial mapping database. This database maps network addresses to the phone-book entry that can establish a connection to the address.
- Get RAS information, including information about existing RAS connections, information about the RAS-capable devices configured on the local computer, and notifications when a RAS connection begins or ends.

Microsoft® Windows NT® version 4.0 also provides support for RAS server administration and for third-party extensions to RAS server security and connection management. Windows® 95 does not provide RAS server support.

RAS Common Dialog Boxes

Windows NT 4.0 provides a set of functions that display the RAS dialog boxes provided by the system. These functions make it easy for applications to display a familiar user interface so that users can perform RAS tasks. For example, users can establish and monitor connections, or work with phone-book entries. Windows 95 does not currently support these functions.

The **RasPhonebookDlg** function displays the main **Dial-Up Networking** dialog box. From this dialog box, the user can dial, edit, or delete a selected phone-book entry, create a new phone-book entry, or specify user preferences. The **RasPhonebookDlg** function uses the **RASPBDLG** structure to specify additional input and output parameters. For example, you can set members of the structure to control the position of the dialog box on the screen. You can use the **RASPBDLG** structure to specify a **RasPBDlgFunc** callback function that receives notifications of user activity while the dialog box is open. For example, RAS calls your **RasPBDlgFunc** function if the user dials, edits, creates, or deletes a phone-book entry.

You can use the **RasDialDlg** function to start a RAS connection operation without displaying the main **Dial-Up Networking** dialog box. With **RasDialDlg**, you specify a phone number or phone-book entry to call. The function displays a stream of dialog boxes that indicate the state of the connection operation. The **RasDialDlg** function uses a **RASDIALDLG** structure to specify additional input and output parameters, such as position of the dialog box and the phone-book subentry to call.

To display the **Dial-Up Networking Monitor** property sheet, call the **RasMonitorDlg** function. This dialog box enables the user to monitor the status of existing connections. The **RasMonitorDlg** function uses a **RASMONITORDLG** structure to specify additional input and output parameters, such as the position of the dialog box and the property sheet page to display on top.

You can call the **RasEntryDlg** function to display a property sheet for creating, editing, or copying a phone-book entry. The **RasEntryDlg** function uses a **RASENTRYDLG** structure to specify additional input and output parameters, such as the position of the dialog box and the type of phone book operation.

RAS Connection Operations

Windows NT 4.0 and later versions provide the **RasPhonebookDlg** and **RasDialDlg** functions that display the built-in user interface for starting a RAS connection operation. For most applications, this is the preferred way to start a RAS connection operation. Windows 95 does not currently support these functions.

The remainder of this section describes the low-level functions for starting a RAS connection. These functions are available on both Windows NT 4.0 (and later versions), and Windows 95.

A RAS client application uses the **RasDial** function to establish a connection to a RAS server. The **RasDial** function starts the connection operation, which is then carried out by the Remote Access Connection Manager.

The Remote Access Connection Manager is a service that handles the details of establishing the connection to the remote server. This service also provides the client with status information during the connection operation. The Remote Access Connection Manager starts automatically when an application loads the RASAPI32.DLL.

The **RasDial** call specifies the following information when it starts a connection operation:

- The connection information that the Remote Access Connection Manager needs to establish the connection.
- An optional notification handler that receives progress notifications during the connection operation. If the **RasDial** call specifies a notification handler, the call is asynchronous; otherwise, it is synchronous.

- An optional **RASDIALEXTENSIONS** structure to enable or disable extensions to the **RasDial** operation. The extensions permit a RAS client to directly enable some modem settings, to control whether RAS uses the prefixes and suffixes in a phone-book entry, and to support paused states during the connection operation.

Synchronous Operations

When **RasDial** is invoked as a synchronous operation, the function does not return until the connection has been established or an error occurs. Synchronous mode provides a simple way for a RAS client to establish a connection. The client can simply call **RasDial**, wait for the function to return, and then call the **RasGetConnectStatus** function to determine whether the connection operation was successful. Once the connection has been established, the client application can terminate without breaking the connection. If an error occurs, the client application must shut down the connection operation before terminating.

The disadvantage of synchronous mode is that the client does not receive progress notifications as the connection operation proceeds. As a workaround for this lack of progress notifications, a synchronous mode client can use a separate thread that calls **RasGetConnectStatus** to poll for and display the current state. However, for RAS clients that want to receive progress information, the preferred technique is to invoke **RasDial** asynchronously.

Asynchronous Operations

When **RasDial** is invoked as an asynchronous operation, the function returns immediately. In asynchronous mode, the **RasDial** call must specify a notification handler that the Remote Access Connection Manager uses to inform the client whenever the connection operation changes states or an error occurs.

The notification handler can be a window to receive messages, or a **RasDialFunc**, **RasDialFunc1**, or **RasDialFunc2** callback function. The Remote Access Connection Manager makes its asynchronous notifications in the context of the thread that made the **RasDial** call. For this reason, the calling thread must not terminate until the connection operation has been successfully established or an error occurs. As in synchronous mode, the client application can safely terminate once the connection has been established, and it must shut down the connection operation if an error occurs.

Phone-Book Files and Connection Information

A **RasDial** call must specify the information that the Remote Access Connection Manager needs to establish the connection. Typically, the **RasDial** call provides the connection information by specifying a phone-book entry. The connection information in a phone-book entry includes phone numbers, bps rates, user authentication information, and other connection information.

A RAS client uses the parameters of the **RasDial** function to specify a phone-book file and an entry in that file. The *lpszPhonebookPath* parameter can specify the name of a phone-book file, or it can be NULL to indicate that the default phone-book file should be used. The *lpRasDialParams* parameter points to a **RASDIALPARAMS** structure that specifies the name of the phone-book entry to use.

To display a list of phone-book entries from which the user can select a connection, a RAS client can call the **RasEnumEntries** function to enumerate the entries in a phone-book file.

To make a connection without using a phone-book entry, the **RasDial** call can specify an empty string for the **szEntryName** member of the **RASDIALPARAMS** structure. The **RASDIALPARAMS.szPhoneNumber** member must contain the number to call. In this case, the Remote Access Connection Manager uses the first available modem port and default values for all other settings.

User Authentication Information

The Remote Access Connection Manager service on the client computer sends a user name and password to the RAS server on the remote computer. Before it will establish a connection, the remote server uses this information to authenticate the user. By default, the Remote Access Connection Manager sends the user name and password of the currently logged-on user. The RAS client can use the **RASDIALPARAMS** structure specified in the **RasDial** call to specify a different user name and password.

If the remote server cannot authenticate the user with the specified information, it can allow the connection operation to enter a paused state to enable the RAS client to collect different authentication data from the user.

Other Connection Information

The members of the **RASDIALPARAMS** structure can also specify the following connection information:

- A phone number to override the number in the phone-book entry.
- A callback phone number that the remote server can call back to establish the connection.
- The name of the remote network domain on which the authentication is to occur.

For the callback number and the domain, the **RASDIALPARAMS** members can either indicate that RAS should use the information in the phone-book entry, or it can provide information that overrides the phone-book data.

A RAS client can use the *lpRasDialExtensions* parameter of the **RasDial** function to control whether RAS uses a phone number prefix or suffix specified in a phone-book entry.

Connection States

During the process of connecting to a remote server, the Remote Access Connection Manager and the RAS server on the remote computer perform several steps to establish the connection. Each of these steps is identified by a connection state. The **RASCONNSTATE** enumeration is a set of values that correspond to these connection states. The connection states can be divided into the following three groups:

Running states
: The running states are the parts of the connection operation that RAS handles automatically, such as connecting to the necessary devices, authenticating the user, and waiting for a callback from the remote server. Unless an error occurs, the RAS client need take no action other than to pass the notification on to the user.

Paused states
: The paused states occur when the remote server pauses the connection operation to get additional input from the user. During a paused state, the user can type a callback number, a different user name and password if the user authentication fails, or a new password if the old one has expired.

Terminal states
: The terminal states occur when the connection has been successfully established, the connection operation has failed, or the connection has been broken by a **RasHangUp** call.

There are several mechanisms that a RAS client can use to determine the current state of a connection operation. When a RAS client executes the **RasDial** function asynchronously, the Remote Access Connection Manager sends progress notifications to the client's notification handler whenever the connection state changes. In addition, the client can use the **RasGetConnectStatus** function to get the current state of any RAS connection operation.

Notification Handlers

An asynchronous **RasDial** call must specify a notification handler. During an asynchronous connection operation, the Remote Access Connection Manager uses the notification handler to inform the RAS client whenever the connection state changes or an error occurs.

The actions performed by a notification handler can be divided into the following categories:

- Handling errors.
- Providing feedback to the user as the connection operation proceeds through the various connection states. See Informational Notifications.
- Handling paused states.
- Signaling the RAS client application when the connection operation has been completed. See *Completion Notifications*.

There are three types of notification handlers, each of which receives the same basic information: the current connection state and an error code that is nonzero only if an error has occurred.

Value	Definition
RasDialFunc	A callback function prototype that receives only the current connection state and error code information.
RasDialFunc1	A callback function prototype that receives the **HRASCONN** connection handle and extended error information in addition to the basic information. The connection handle parameter makes **RasDialFunc1** useful for client applications that support multiple simultaneous connection operations. This allows the client to specify the same callback function for all operations, and enables the callback function to determine which connection is changing states.
RasDialFunc2	A callback function similar to **RasDialFunc1**. However, **RasDialFunc2** is enhanced to support multilink connections.
Window handle	A window handle to which RAS sends WM_RASDIALEVENT messages containing the current connection state and error code information. Use this method if your source code must be compatible with 16-bit Windows, because 16-bit Windows does not support either of the callback functions.

The Remote Access Connection Manager suspends the connection operation until the notification handler returns. For this reason, the handler should return as soon as possible unless an error has occurred.

The **RasDial** function should not be called from within a notification handler. The other remote access functions (**RasGetConnectStatus**, **RasEnumEntries**, **RasEnumConnections**, **RasGetErrorString**, and **RasHangUp**) can be called from within a handler.

Handling RAS Errors

When an error occurs, the Remote Access Connection Manager invokes the client's notification handler. The notification indicates the connection state when the error occurred, and a code that identifies the error. In these cases, the notification handler should call **RasHangUp** to end the RAS connection.

The RAS client can use the **RasGetErrorString** function to get a display string describing the error.

Informational Notifications

For the connection states known as running states, no action is required of the notification handler unless an error occurs. Running states occur during the parts of the connection operation that RAS handles automatically, such as connecting to the necessary devices, authenticating the user, and waiting for a callback from the remote server. The notification is simply a progress report to the client.

The client can choose to pass these informational notifications on to the user. In some running states, the client may want to display additional information. For example, a notification handler that receives a RASCS_ConnectDevice notification can call the **RasGetConnectStatus** function to get the name and type of the device being connected to. Another example is when the client receives a RASCS_Projected notification. This occurs when the RAS projection phase of the connection operation has been completed. The client can call the **RasGetProjectionInfo** function to get additional information about the projection. The client can use this information to notify the user as to which network protocols can be used by this connection.

You should avoid writing code that depends on the order or occurrence of particular informational states, because this may vary between platforms.

Completion Notifications

The Remote Access Connection Manager continues progress notifications until the connection operation has been completed. This occurs in the following situations:

- The handler receives a RASCS_Connected, or RASCS_Disconnected notification. The RAS client application can exit without breaking any established connection.
- An error occurs. The handler receives a notification indicating the error and the connection state when the error occurred. The RAS client application can exit.

The RAS client application should not assume the connection operation is complete after calling **RasHangUp**. It should wait for one of the preceding conditions before exiting.

Paused States

During a connection operation, there can be times when the remote server cannot proceed without additional information from the local user. Beginning with Microsoft® Windows NT® version 3.5, the **RasDial** function supports paused states. A paused state allows the Remote Access Connection Manager to suspend a connection operation so the RAS client application can collect information from the user.

Paused states are useful in the following situations:

- When the user needs to provide a callback number.
- When the user authentication fails, the user can type in a different user name and password.
- When the user's password has expired, the user can provide a new password.

By default, paused state support is disabled. RAS clients that want to support paused states must set the RDEOPTS_PausedStates flag in the **RASDIALEXTENSIONS** structure passed as a parameter to **RasDial**.

When a paused state occurs, the Remote Access Connection Manager invokes the client's notification handler. If paused state support is disabled, the notification message indicates an error, and the connection operation fails. If it is enabled, the Connection Manager pauses the connection operation to wait for the RAS client's response. The RAS client can resume the connection operation by a second **RasDial** call, or terminate it by calling the **RasHangUp** function.

After getting the user's input, the RAS client restarts the connection operation by calling **RasDial** again. This second **RasDial** call must specify the following information:

- The connection handle that was returned by the original **RasDial** call.
- The same notification handler as the original **RasDial** call.
- The user's input in the appropriate members of the **RASDIALPARAMS** structure. Other members of the **RASDIALPARAMS** structure should have the same information as specified in the original **RasDial** call.

The second **RasDial** call cannot be made from within the notification handler.

Callback Connections

RAS supports connections in which the remote server hangs up and then calls back to the client to establish the connection.

For each user that can connect to a RAS server, the server stores a callback attribute that controls how the connection is made. The default attribute is No Callback, which means that the user can connect to the RAS server without a callback. Alternatively, the administrator of the RAS server can assign to a user either the Preset or Set-By-Caller callback attribute.

For a user assigned the Preset restriction, the administrator specifies a phone number that the RAS server must call back to establish a connection. The user cannot specify a different number, and the connection cannot be made without a callback.

A Preset callback operation is handled automatically by the Remote Access Connection Manager and the remote server. The RAS client application does not need to do anything other than provide feedback to the user when the notification handler is called during the various states of the callback operation.

A user assigned the Set By Caller privilege can choose to connect either with or without a callback. The **RasDial** call uses the **szCallBackNumber** member of the **RASDIALPARAMS** structure to indicate the choice.

The **szCallBackNumber** member can simply specify the callback number; or, to establish the connection without a callback, **szCallBackNumber** can point to an empty string, "". In either of these cases, the Remote Access Connection Manager handles the connection operation automatically. As with a Preset callback operation, the RAS client does not need to perform any action other than to provide feedback to the user.

If the **RasDial** call enables paused states, **szCallBackNumber** can point to an asterisk string, "*", to indicate that the connection operation should enter a paused state to allow the user to type in the callback number. In this case, the connection operation for a Set By Caller user enters a paused state after the remote server has authenticated the user. During the paused state, the RAS client gets the callback number input from the user. The client then resumes the connection operation by making a second **RasDial** call in which **szCallBackNumber** specifies the number supplied by the user.

Note If paused states are not enabled there is a different meaning when **szCallBackNumber** points to an asterisk string, "*". In this case, the asterisk indicates that the callback number is stored in the phone-book file specified by the **RasDial** call.

Disconnecting

When a RAS client application starts a connection operation, the **RasDial** call receives an **HRASCONN** connection handle to identify the connection. If the returned handle is not NULL, the client must eventually call the **RasHangUp** function to end the connection. If an error occurs during the connection operation, the client must call **RasHangUp** even though the connection was never established.

The application that calls **RasHangUp** should not exit immediately, because the Remote Access Connection Manager needs time to properly terminate the connection. Instead, the application should wait until the **RasGetConnectStatus** function returns ERROR_INVALID_HANDLE, indicating that the connection has been deleted.

A RAS client application might need to end a connection even though it does not have the handle returned by **RasDial**. For example, the application that called **RasDial** might have exited once the connection was successfully established. In this case, the disconnecting application can use the **RasEnumConnections** function to get all the current connections. For each connection, **RasEnumConnections** returns a **RASCONN** structure containing the **HRASCONN** connection handle and the phone-book entry name or phone number specified when the connection operation was started. This information can be used to display a list of connections from which the user can select the connection to end.

RAS Custom Scripting

Developers can create a custom-scripting DLL that resides on a RAS client computer. This DLL can communicate with the server during the process of establishing a connection.

Setting Up the DLL

To set up the DLL, create a value with the name **CustomScriptDllPath** under the following registry key:

\\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Rasman\Parameters

This value should be of type **REG_EXPAND_SZ**. The value should contain the path to the custom-scripting DLL. Only one custom-scripting DLL is supported for each RAS client computer.

Configuring the Phone-Book Entries

RAS will invoke **RasCustomScriptExecute** for a connection only if the phone-book entry for the connection has the RASEO_CustomScript option set. See the **dwfOptions** member of **RASENTRY** for a description of phone-book entry options. Use the **RasGetEntryProperties** and **RasSetEntryProperties** functions to set this option programmatically.

Interaction Between the Server, RAS, and the Custom-Scripting DLL

The custom scripting DLL should export a single entry point: **RasCustomScriptExecute**. RAS will call this function during the RASCS_Interactive state of the connection process. The RASCS_Interactive state is a paused state, which allows the user to interact with a user interface presented by the custom-scripting DLL. See **RASCONNSTATE** for more information about connection states.

RAS will pass as parameters to the **RasCustomScriptExecute** function:

- A handle to the port on the client computer that is being used for the connection.
- Strings that identify the phone book and entry for the connection.
- RAS also passes in a handle to a window to enable the DLL to present a user interface.
- A set of function pointers that the DLL can use to communicate with the server.

See **RasCustomScriptExecute** for more information about these parameters.

RAS mediates the dialog between the server and the custom-scripting DLL. Typically, the server initiates the dialog. For example, the server may request the user name and password of the user.

RAS makes no assumptions about the type of server to which the client is connected. The server need not use Windows NT version 4.0 or Windows 2000.

RAS Phone Books

Phone books provide a standard way to collect and specify the information that the Remote Access Connection Manager needs to establish a remote connection. Phone books associate entry names with information such as phone numbers, COM ports, and modem settings. Each phone-book entry contains the information needed to establish a RAS connection.

Windows NT/2000: Phone books are stored in phone-book files, which are text files that contain the entry names and associated information. RAS creates a phone-book file called RASPHONE.PBK. The user can use the main **Dial-Up Networking** dialog box to create personal phone-book files. The Win32 API does not currently provide support for creating a phone-book file. Some RAS functions, such as the **RasDial** function, have a parameter that specifies a phone-book file. If the caller does not specify a phone-book file, the function uses the default phone-book file, which is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

Windows NT version 4.0 provides the **RasPhonebookDlg** and **RasEntryDlg** functions that display the built-in RAS user interface that enable users to work with phone books and phone-book entries.

Windows 95: Dial-up networking stores phone-book entries in the registry rather than in a phone-book file. Windows 95 does not support personal phone-book files. Windows 95 does not support the functions that display the built-in RAS dialog boxes.

Phone-Book Entries

Phone-book entries contain the information necessary to establish a RAS connection. A user or administrator can use the **Dial-Up Networking** dialog box to create, edit, and dial phone-book entries.

Windows 95: Windows 95 supports a limited set of the Win32 functions for working with phone-book entries. You can use the **RasCreatePhonebookEntry** and **RasEditPhonebookEntry** functions to create or edit a phone-book entry. These functions display a dialog box in which the user can specify information about the phone-book entry. You can use the **RasGetEntryDialParams** and **RasSetEntryDialParams** functions to set or retrieve the connection parameters for a phone-book entry. The **RasEnumEntries** function retrieves an array of **RASENTRYNAME** structures that contain the phone-book entry names.

Windows NT version 4.0 supports the functions described for Windows 95, as well as a number of additional functions that an application can use to work with phone books and phone-book entries.

The **RasEntryDlg** function displays a property sheet that enables the user to create, edit, or copy phone-book entries. The **RasCreatePhonebookEntry** and **RasEditPhonebookEntry** functions call the **RasEntryDlg** function. You can use the

RasRenameEntry function to rename a phone-book entry, or the **RasDeleteEntry** to delete an entry. The **RasValidateEntryName** determines whether a specified string has the correct format to be used as an entry name.

You can use the **RasGetEntryProperties** and **RasSetEntryProperties** to get and set additional information about a phone-book entry. These functions use a **RASENTRY** structure.

The **RasGetCredentials** and **RasSetCredentials** functions get and set the user credentials associated with a specified RAS phone-book entry. These functions use a **RASCREDENTIALS** structure.

The **RasGetCountryInfo** function retrieves country-specific dialing information from the Windows Telephony list of countries. **RasGetCountryInfo** uses the **RASCTRYINFO** structure.

Subentries and Multilink Connections

Windows NT version 4.0 provides support for phone-book subentries, which enable multilink connections. A multilink connection combines the bandwidth of multiple connections to provide a single connection with higher bandwidth.

A RAS phone-book entry can have zero or more subentries. The **RasGetEntryProperties** function retrieves a **RASENTRY** structure that includes information about the subentries of a phone-book entry. The **dwSubEntries** member of the **RASENTRY** structure indicates the number of subentries. Phone-book entries initially have no subentries. To add subentries to a phone-book entry, use the **RasSetSubEntryProperties** function.

The properties for each subentry include a phone number and the name and type of the TAPI device to use when dialing the subentry. In addition, a subentry can include a list of alternate phone numbers to dial if RAS cannot make a connection using the primary number. The **RasSetSubEntryProperties** and **RasGetSubEntryProperties** functions use the **RASSUBENTRY** structure to set and retrieve the properties of a specified phone-book subentry. Subentries are identified by a one-based index.

You can call the **RasSetEntryProperties** function to configure a multilink RAS entry to connect all subentries when it is first dialed. Alternatively, you can configure an entry to provide variable bandwidth. In this case, RAS connects a single subentry initially, and then connects or disconnects additional subentries as needed. For a variable-bandwidth multilink connection, you can use the **RASDIALPARAMS** structure to specify the initial subentry to connect when you call the **RasDial** function. When using the **RasDialDlg** function to connect a multilink entry, you can use the **RASDIALDLG** structure to specify the initial subentry to connect.

For a variable-bandwidth multilink connection, use the **RASENTRY** structure with the **RasSetEntryProperties** function to specify the parameters for connecting and disconnecting the individual subentries. RAS connects an additional subentry when the bandwidth being used exceeds a specified percentage of the available bandwidth for a specified interval.

If you call the **RasDial** function to establish a multilink connection, you can specify a **RasDialFunc2** callback function to receive notifications about the connection. **RasDialFunc2** is similar to the **RasDialFunc1** callback function, except that it provides additional information for a multilink connection, such as the index of the subentry that caused the notification. RAS calls your **RasDialFunc2** function when it connects or disconnects a subentry.

You can use an **HRASCONN** connection handle to hang up or retrieve information about a multilink connection. You can get a connection handle for each of the subentry connections that make up the multilink, as well as for the combined multilink connection. When you call the **RasDial** function to establish a multilink connection, **RasDial** returns a handle to the combined multilink connection. Similarly, **RasEnumConnections** returns the combined multilink handle when you enumerate connections. To get a handle to one of the subentry connections in a multilink connection, call the **RasGetSubEntryHandle** function.

You can use the combined multilink connection handle and the subentry connection handles in the **RasHangUp**, **RasGetConnectStatus**, and **RasGetProjectionInfo** functions. Calling **RasHangUp** with a combined multilink handle terminates the entire connection; calling it with a subentry handle hangs up only that subentry connection. Similarly, **RasGetConnectStatus** returns information for the combined or individual connection, depending on the handle specified. The projection information returned by **RasGetProjectionInfo** for a multilink entry is the same for each of the subentry connection handles as it is for the main connection handle.

RAS AutoDial

Windows NT version 4.0 supports a feature known as AutoDial. Windows 95 and Windows NT version 3.51 and earlier do not support the AutoDial feature.

When an attempt to connect to a network address fails because the host cannot be reached, the AutoDial feature can automatically start a dial-up connection operation. To do this, AutoDial searches its database of network addresses to find a phone-book entry that it can use to establish the connection.

AutoDial Mapping Database

The AutoDial mapping database maps network addresses to RAS phone-book entries. The database can include IP addresses (for example, "127.95.1.4"), Internet host names (for example, "*www.microsoft.com*"), or NetBIOS names (for example, "products1"). Associated with each address in the AutoDial database is a set of one or more **RASAUTODIALENTRY** entries. Each of these entries specifies a phone-book entry that RAS can dial to connect to the address from a particular Telephony Application Programming Interface (TAPI) dialing location. For more information about TAPI dialing locations, see the *TAPI documentation.*

AutoDial automatically creates entries in the AutoDial mapping database in two situations:

- **When an attempt to connect to a network address fails**

  If there is no entry for the address in the mapping database, and the computer is not connected to a network (either directly or through RAS), AutoDial prompts the user to specify the information necessary to establish a dial-up connection. If the user provides the information and the dial-up connection operation is successful, AutoDial stores the information in the mapping database.

- **When the computer is connected to a network through RAS**

  Whenever the user connects to a network address, AutoDial creates an entry in the database. The entry maps the network address to the phone-book entry that was used to establish the RAS connection.

You can use the **RasSetAutodialAddress** function to add an address to the AutoDial mapping database, delete an address from the database, or change the AutoDial entries associated with an existing address in the database. You can use the **RasGetAutodialAddress** function to retrieve the AutoDial entries associated with a specified network address in the AutoDial mapping database. The **RasEnumAutodialAddresses** function returns a list of all addresses in the AutoDial mapping database.

AutoDial Connection Operations

When an attempt to connect to a network address fails because the host cannot be reached, the system searches the AutoDial mapping database for the address. If the address is in the database, the system initiates an AutoDial operation for the **RASAUTODIALENTRY**, if any, that corresponds to the local TAPI dialing location.

The Win32 API provides functions that enable you to set and query AutoDial parameters that control AutoDial connections. You can call the **RasSetAutodialEnable** function to enable or disable the AutoDial feature for a specified TAPI dialing location. The **RasGetAutodialEnable** function indicates whether the AutoDial feature is enabled for a specified TAPI dialing location. For more information about TAPI dialing locations, see the TAPI documentation. You can call the **RasSetAutodialParam** function to set other AutoDial connection parameters. For example, you can disable AutoDial connections for the current logon session. Call the **RasGetAutodialParam** function to determine the current value of the AutoDial connection parameters.

The system provides a default user interface for AutoDial dialing operations. However, you can create an AutoDial dynamic-link library (DLL) to provide a custom user interface for AutoDial dialing operations involving specified phone-book entries. Your AutoDial DLL must export both an ANSI and a Unicode version of a **RASADFunc** AutoDial handler.

To enable your custom AutoDial handler for a phone-book entry, call the **RasSetEntryProperties** function to set the properties for that entry. The **szAutodialDll** and **szAutodialFunc** members of the **RASENTRY** structure passed to **RasSetEntryProperties** specify the name of your AutoDial DLL and the name of your **RASADFunc** function, excluding the "A" or "W" suffix.

When the system starts an AutoDial operation for a phone-book entry with a custom AutoDial handler, it calls the specified **RASADFunc**. The RASADFunc function receives a pointer to a **RASADPARAMS** structure that indicates the location and parent window for the window of your user interface. Your **RASADFunc** can start a thread to perform the custom dialing operation. The **RASADFunc** function returns TRUE to indicate that it took over the dialing, or FALSE to allow the system to perform the dialing. Your custom dialing operation must use the **RasDial** function to do the actual dialing. When the dialing operation has been completed, the custom dialing operation indicates success or failure by setting the variable pointed to by the *lpdwRetCode* parameter passed to **RASADFunc**.

RAS Configuration and Connection Information

Applications running on Windows NT version 4.0 and later versions, and Windows 95, can use the **RasEnumConnections** function to get information about the existing connections on the local computer. The information for each connection includes a connection handle and the name of the phone-book entry used to establish the connection. You can use the connection handle in a call to the **RasGetConnectStatus** function get the current status of the connection.

Windows NT 4.0 and later versions provide two new functions for retrieving RAS information. Windows 95 does not support these functions.

The **RasEnumDevices** function returns the name and type of the RAS-capable devices that are configured on the local computer.

The **RasConnectionNotification** function specifies an event object that the system signals when a RAS connection is created or terminated.

RAS Server Administration

Windows NT version 4.0 provides a set of functions for administering user permissions and ports on Windows NT/Windows 2000 RAS servers. Windows 95 does not support these functions. Using these functions, you can develop a RAS server administration application to perform the following tasks:

- Enumerate those users who have a specified set of RAS permissions
- Assign or revoke RAS permissions for a specified user
- Enumerate the configured ports on a RAS server
- Get information and statistics about a specified port on a RAS server

- Reset the statistics counters for a specified port
- Disconnect a specified port

You can also install a RAS server administration DLL for auditing user connections and assigning IP addresses to dial-in users. The DLL exports a set of functions that the RAS server calls whenever a user tries to connect or disconnect.

RAS User Account Administration

A Windows NT version 4.0 RAS server uses a user account database that contains information about a set of user accounts. The information includes a user's RAS privileges, which are a set of bit flags that determine how the RAS server responds when the user calls to connect. The RAS server administration functions enable you to locate the user account database, and to get and set the RAS privileges for user accounts.

A Windows NT version 4.0 RAS server can be part of a Windows NT/Windows 2000 domain, or it can be a stand-alone Windows NT Server or Workstation that is not part of a domain. For a server that is part of a domain, the user account database is stored on the Windows NT/Windows 2000 server that is the Primary Domain Controller (PDC). A stand-alone server stores its own local user account database. To get the name of the server that stores the user account database used by a specified RAS server, you can call the **RasAdminGetUserAccountServer** function. You can then use the name of the user account server in a call to the **NetQueryDisplayInformation** function to enumerate the users in a user account database. You can also use the server name in calls to the **RasAdminUserGetInfo** and **RasAdminUserSetInfo** functions to get and set the RAS privileges for a specified user account.

The **RasAdminUserGetInfo** and **RasAdminUserSetInfo** functions use the **RAS_USER_0** structure to specify a user's RAS privileges and call-back phone number. The RAS privileges indicate the following information:

- Whether the user can make a remote connection to the server or the domain to which the server belongs.
- Whether the user can establish a connection through a call-back, in which the RAS server hangs up and then calls back to the user to establish the connection.

Each user account specifies one of the following flags to indicate the user's call-back privilege.

Value	Meaning
RASPRIV_NoCallback	The RAS server will not call back the user to establish a connection.
RASPRIV_AdminSetCallback	When the user calls, the RAS server hangs up and calls a preset call-back phone number stored in the user account database. The **szPhoneNumber** member of the **RAS_USER_0** structure contains the user's call-back phone number.
RASPRIV_CallerSetCallback	When the user calls, the RAS server provides the option of specifying a call-back phone number. The user can also choose to connect immediately without a call back. The **szPhoneNumber** member contains a default number that the user can override.

RAS Server and Port Administration

The RAS server administration functions enable you to get information about a specified RAS server and its ports. These functions also enable you to terminate a connection on a specified RAS server port.

The **RasAdminServerGetInfo** function returns a **RAS_SERVER_0** structure that contains information about the configuration of a RAS server. The returned information includes the number of ports currently available for connection, the number of ports currently in use, and the server version number.

The **RasAdminPortEnum** function retrieves an array of **RAS_PORT_0** structures that contains information for each of the ports configured on a RAS server. The information for each port includes:

- The name of the port
- Information about the device attached to the port
- Whether the RAS server associated with the port is a Windows NT/Windows 2000 Server
- Whether the port is currently in use, and if it is, information about the connection

You can call the **RasAdminPortGetInfo** function to get additional information about a specified port on a RAS server. This function returns a **RAS_PORT_1** structure that contains a **RAS_PORT_0** structure and additional information about the current state of the port. The **RasAdminPortGetInfo** function also returns an array of **RAS_PARAMETERS** structures that describe the values of any media-specific keys associated with the port. A **RAS_PARAMETERS** structure uses a value from the **RAS_PARAMS_FORMAT** enumeration to indicate the format of the value for each media-specific key.

The **RasAdminPortGetInfo** function also returns a **RAS_PORT_STATISTICS** structure that contains various statistic counters for the current connection, if any, on the port. For a port that is part of a multilink connection, **RasAdminPortGetInfo** returns statistics for the individual port and cumulative statistics for all ports involved in the connection. You can use the **RasAdminPortClearStatistics** function to reset the statistic counters for the port. The **RasAdminPortDisconnect** function disconnects a port that is in use.

Use the **RasAdminFreeBuffer** function to free memory allocated by the **RasAdminPortEnum** and **RasAdminPortGetInfo** functions. Use the **RasAdminGetErrorString** function to get a string that describes a RAS error code returned by one of the RAS Server Administration (RasAdmin) functions.

RAS Administration DLL

Windows NT version 4.0 enables you to install a RAS administration DLL on a Windows NT version 4.0RAS server. The DLL exports functions that the RAS server calls whenever a user tries to connect or disconnect. You can use the DLL to perform the following administrative functions:

- Decide whether to allow a user to connect to the server. This can provide a security check in addition to the standard RAS user authentication.
- Record the time that each user connects to and disconnects from the server. This can be useful for billing or auditing purposes.
- Assign an IP address to each user. This can be useful for security purposes to map a user's connection to a specific computer.

Implement the following functions when developing a RAS server administration DLL.

- **RasAdminAcceptNewConnection**
- **RasAdminConnectionHangupNotification**
- **RasAdminGetIpAddressForUser**
- **RasAdminReleaseIpAddress**

A RAS administration DLL must implement and export all of the above functions. If any of the functions are not implemented, the remote access service will fail to start.

The **RasAdminAcceptNewConnection** and **RasAdminConnectionHangupNotification** functions enable the DLL to audit user connections to the server. A Windows NT/Windows 2000 RAS server calls the **RasAdminAcceptNewConnection** function whenever a user tries to connect. The function can prevent the user from connecting. You can also use the function to generate an entry in a log for billing or auditing. When the user disconnects, the RAS server calls the **RasAdminConnectionHangupNotification** function, which can log the time at which the user disconnected.

After the RAS server has authenticated a caller, it calls the **RasAdminGetIpAddressForUser** function to get an IP address for the remote client. The DLL can use this function to provide an alternate scheme for mapping an IP address to a dial-in user. If **RasAdminGetIpAddressForUser** is not implemented, a RAS server connects a remote user to an IP address selected from a static pool of IP addresses, or one selected by a Dynamic Host Configuration Protocol (DHCP) server. The **RasAdminGetIpAddressForUser** function allows the DLL to override this default IP address and specify a particular IP address for each user. The **RasAdminGetIpAddressForUser** function can set a flag that causes RAS to call the **RasAdminReleaseIPAddress** function when the user disconnects. The DLL can use **RasAdminReleaseIPAddress** to update its user-to-IP-address map.

RAS serializes calls into the administration DLL. A call into one of the DLL's functions for a given RAS client will never be preempted by a call to that function for a different RAS client; the initial call is guaranteed to be complete before RAS calls the function for the other client. Furthermore, serialization extends to certain groups of functions. The IP address functions are serialized as a group; a call into either **RasAdminGetIpAddressForUser** or **RasAdminReleaseIpAddress** will block calls into both until the initial call is complete. **RasAdminAcceptNewConnection** and **RasAdminConnectionHangupNotification** are also serialized as a group.

RAS executes the functions for assigning IP addresses in one process and executes the functions for connection and disconnection notifications in another process. Consequently, the DLL should not depend on shared data between the two sets of functions.

The RAS server logs an error in the system event log if an error occurs when it tries to load a RAS administration DLL or when calling one of the DLL's functions. This can happen, for example, if the DLL specified the wrong name for an exported function, or if it did not include the function name in the .def file. The entry in the event log indicates the reason for the failure.

Windows 2000 and later: RAS administration DLLs that implement this function interface will not work on Windows 2000 and later versions. Instead, use the MprAdmin function interface provided with the more recent versions of Windows. For more information, see the RAS Administration Reference in the Routing and RAS documentation.

RAS Administration DLL Registry Setup

The setup program for a third-party RAS administration DLL must register the DLL with RAS by providing information under the following key in the registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll

To register the DLL, set the following values under this key.

Value name	Value data
DisplayName	A REG_SZ string that contains the user-friendly display name of the DLL.
DLLPath	A REG_SZ string that contains the full path of the DLL.

For example, the registry entry for a RAS administration DLL from a fictional company named Netwerks Corporation might be:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll
DisplayName : REG_SZ : Netwerks RAS Admin DLL
DLLPath : REG_SZ : C:\nt\system32\ntwkadm.dll

The setup program for a RAS administration DLL should also provide remove/uninstall functionality. If a user removes the DLL, the setup program should delete the DLL's registry entries.

RAS Security Host Support

Windows NT version 4.0 provides a way for a third-party RAS security DLL to enhance the built-in RAS security features. Windows 95 does not provide this support.

The Windows NT/Windows 2000 RAS server provides security mechanisms for validating the network access of remote users. When a RAS server receives a call, it validates the user's credentials against the local or domain account database. RAS also supports call-back security, in which the RAS server hangs up and then calls back to the remote user to establish the connection. For networks in which this level of security is not enough, you can install a third-party RAS security DLL. The security DLL can then authenticate a remote user by reading security information from a database other than the standard Windows NT/Windows 2000 user account database.

When the RAS server receives a call, it invokes the security DLL to authenticate the remote user. The RAS security host support provides a mechanism for the security DLL to communicate with the remote user through a terminal window on the remote computer. In a typical scenario, the security DLL asks for the logon name of the remote user. The DLL then uses its private security database to formulate a challenge to send to the remote terminal. For example, the challenge could be a code that the user must provide as input to a cardkey reader. The cardkey reader then displays a response that the remote user types in the terminal window. The security DLL then validates the response against the user's information in the private security database.

If the security DLL authenticates the remote user, the RAS server performs its own authentication. This ensures that RAS security always authenticates a remote user, even if a security DLL is installed that grants access to all users.

Note Windows NT/Windows 2000 currently provides RAS security host support only for asynchronous connections; other media, such as ISDN, are not supported.

Registering a RAS Security DLL

The setup program for a RAS security DLL must register the DLL with the Windows NT/Windows 2000 RAS server. Only one RAS security DLL can be registered; Windows NT/Windows 2000 does not support multiple security DLLs. To register a RAS security DLL, set the *DLLPath* value under the following key in the registry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\SecurityHost
```

Value Name	Value Data
DLLPath	A REG_SZ string that contains the path of the DLL. This string should specify the full path unless the DLL is in a directory listed in the system path.

The setup program for a RAS security DLL must also provide remove/uninstall functionality. If a user removes the DLL, the setup program must delete the *DLLPath* value from the registry. The RAS service will not start if the *DLLPath* value specifies a DLL that cannot be found.

A RAS security DLL must export the **RasSecurityDialogBegin** and **RasSecurityDialogEnd** functions.

RAS Server Security Authentication

When a Windows NT/Windows 2000 RAS server receives a call, it invokes the **RasSecurityDialogBegin** function of the registered RAS security DLL, if there is one. This call notifies the security DLL to begin its authentication of the remote user. The RAS server calls **RasSecurityDialogBegin** before performing its PPP or RAS authentication.

The **RasSecurityDialogBegin** call passes the following information to the security DLL:

- A port handle to identify the connection
- Pointers to buffers to use when communicating with the remote user
- A pointer to a **RasSecurityDialogComplete** function to call when the authentication has been completed

The port handle and buffer pointers are valid until the security DLL calls **RasSecurityDialogComplete** to terminate the authentication transaction.

The **RasSecurityDialogComplete** notifies the RAS server of the results of the security DLL's authentication of the remote user. If the security DLL reports success, the RAS server proceeds with its PPP and RAS authentication of the remote user. If the security DLL reports that the remote user failed the authentication, or that an error occurred, the RAS server hangs up and logs the error or failed authentication in the Windows NT/Windows 2000 event log.

RAS Security DLL Authentication Transaction

The Windows NT/Windows 2000 RAS server calls the security DLL's **RasSecurityDialogBegin** function to begin an authentication of a remote user. The RAS server is blocked and cannot accept any other calls until **RasSecurityDialogBegin** returns. For this reason, **RasSecurityDialogBegin** should copy the input parameters, create a thread to perform the authentication, and return as quickly as possible.

The thread created by the security DLL uses the **RasSecurityDialogSend** and **RasSecurityDialogReceive** functions to communicate with the remote computer. These functions are not available for static import from any library. Instead, the security DLL must use the **LoadLibrary** and **GetProcAddress** functions to dynamically link to these functions in RASMAN.DLL.

During an authentication transaction, the RAS connection manager on the remote computer displays a terminal window. The thread of the security DLL calls **RasSecurityDialogSend** to send a message to display in the terminal window. The thread then calls **RasSecurityDialogReceive** to receive the input that the remote user types in the terminal window. The thread can make any number of **RasSecurityDialogSend** calls, with each call followed by a **RasSecurityDialogReceive** call. After each call to **RasSecurityDialogReceive**, the thread must call one of the wait functions, such as **WaitForSingleObject**, to wait for the asynchronous send and receive operations to be completed. The RAS server signals an event object when the receive operation has been completed or when an optional time-out interval has elapsed.

When the thread has finished authenticating the remote user, it calls the **RasSecurityDialogComplete** function. This call passes a **SECURITY_MESSAGE** structure containing the results of the authentication transaction to the RAS server. The RAS server then performs a cleanup sequence that includes a call to the DLL's **RasSecurityDialogEnd** function. This gives the security DLL an opportunity to perform any necessary cleanup.

The security DLL can call the **RasSecurityDialogGetInfo** function to retrieve information about the port associated with an authentication transaction. **RasSecurityDialogGetInfo** fills in a **RAS_SECURITY_INFO** structure that indicates the state of the last **RasSecurityDialogReceive** call for the port.

Using Remote Access Service

The following section explains how to use Remote Access Service features in an application.

Linking to the Remote Access DLL

If an application links statically to the RASAPI32 DLL, the application will fail to load if Remote Access Service is not installed. A RAS application can load when RAS is not installed by using **LoadLibrary** to load the DLL, and **GetProcAddress** to obtain pointers to the RAS functions.

The Win32 RAS functions are in RASAPI32.DLL. The import library for these functions is RASAPI32.LIB. To use the RAS functions, your programs must include the following files.

File	Description
RAS.H	Contains the RAS function prototypes, constants, and structure definitions.
RASERROR.H	Contains the RAS error codes.

CHAPTER 7

RAS Functions

Use the following functions to implement RAS functionality:

ORASADFunc
RASADFunc
RasClearConnectionStatistics
RasClearLinkStatistics
RasConnectionNotification
RasCreatePhonebookEntry
RasCustomDeleteEntryNotify
RasCustomDial
RasCustomDialDlg
RasCustomEntryDlg
RasCustomHangUp
RasDeleteEntry
RasDial
RasDialDlg
RasDialFunc
RasDialFunc1
RasDialFunc2
RasEditPhonebookEntry
RasEntryDlg
RasEnumAutodialAddresses
RasEnumConnections
RasEnumDevices
RasEnumEntries
RasFreeEapUserIdentity
RasGetAutodialAddress
RasGetAutodialEnable
RasGetAutodialParam
RasGetConnectionStatistics
RasGetConnectStatus
RasGetCountryInfo
RasGetCredentials
RasGetCustomAuthData
RasGetEapUserData
RasGetEapUserIdentity
RasGetEntryDialParams
RasGetEntryProperties
RasGetErrorString
RasGetLinkStatistics
RasGetProjectionInfo
RasGetSubEntryHandle
RasGetSubEntryProperties
RasHangUp
RasInvokeEapUI
RasMonitorDlg
RasPBDlgFunc
RasPhonebookDlg
RasRenameEntry
RasSetAutodialAddress
RasSetAutodialEnable
RasSetAutodialParam
RasSetCredentials
RasSetCustomAuthData
RasSetEapUserData
RasSetEntryDialParams
RasSetEntryProperties
RasSetSubEntryProperties
RasValidateEntryName

ORASADFunc

The **ORASADFunc** function is an application-defined callback function that you can use to provide a customized user interface for autodialing.

This prototype is provided for compatibility with earlier versions of Windows. New applications should use the **RASADFunc** callback function. Support for this prototype may be removed in future versions of RAS.

```
BOOL WINAPI ORASADFunc(
  HWND hwndOwner,         // handle of an owner window
  LPSTR lpszEntry,        // pointer to a phone book entry
  DWORD dwFlags,          // reserved; must be zero
  LPDWORD lpdwRetCode     // receives the results of a
                          //dialing operation
);
```

Parameters

hwndOwner
Handle of the owner window.

lpszEntry
Pointer to a null-terminated string that specifies the phone book entry to use.

dwFlags
Reserved; must be zero.

lpdwRetCode
Pointer to a variable that the callback function fills in with the results of the dialing operation. If the dialing operation succeeds, set this variable to ERROR_SUCCESS. If the dialing operation fails, set it to a nonzero value.

Return Values

If the callback function performs the dialing operation, return TRUE. Use the *lpdwRetCode* parameter to indicate the results of the dialing operation.

If the callback function does not perform the dialing operation, return FALSE. In this case, the system uses the default user interface for dialing.

Remarks

If your **ORASADFunc** function performs the dialing operation, it presents its own user interface for dialing and calls the **RasDial** function to do the actual dialing. Your **ORASADFunc** then returns TRUE to indicate that it took over the dialing. When the dialing operation has been completed, set the variable pointed to by *lpdwRetCode* to indicate success or failure.

To enable an **ORASADFunc** handler for a phone book entry, use the **RASENTRY** structure in a call to the **RasSetEntryProperties** function. The **szAutodialDll** member specifies the name of the DLL that contains the handler, and the **szAutodialFunc** member specifies the exported name of the handler.

The **ORASADFunc** function is a placeholder for the library-defined function name. The **ORASADFUNC** type is a pointer to an **ORASADFunc** function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASADFunc**, **RasDial**, **RASENTRY**, **RasSetEntryProperties**

RASADFunc

The **RASADFunc** function is an application-defined callback function that you can use to provide a customized user interface for autodialing.

```
BOOL WINAPI RASADFunc(
  LPTSTR lpszPhonebook,   // pointer to full path and file
                          // name of phone book file
  LPTSTR lpszEntry,       // pointer to the entry name
                          // to validate
  LPRASADPARAMS lpAutoDialParams,
                          // pointer to a RASADPARAMS
                          //structure
  LPDWORD lpdwRetCode     // receives results of
                          //dialing operation
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry

Pointer to a null-terminated string that specifies the phone book entry to use.

lpAutoDialParams

Pointer to a **RASADPARAMS** structure that indicates how to position the window of your AutoDial user interface. The structure may also specify a parent window for your AutoDial window.

lpdwRetCode

Pointer to a variable in which you must return a value if you perform the dialing operation. If the dialing operation succeeds, set this variable to ERROR_SUCCESS. If the dialing operation fails, set it to a nonzero value.

Return Values

If your application performs the dialing operation, return TRUE. Use the *lpdwRetCode* parameter to indicate the results of the dialing operation.

If your application does not perform the dialing operation, return FALSE. In this case, the system uses the default user interface for dialing.

Remarks

When the system starts an AutoDial operation for a phone book entry with a custom AutoDial handler, it calls the specified **RASADFunc**. Your **RASADFunc** can start a thread to perform the custom-dialing operation. The **RASADFunc** function returns TRUE to indicate that it took over the dialing, or FALSE to allow the system to perform the dialing.

If your **RASADFunc** function performs the dialing operation, it presents its own user interface for dialing and calls the **RasDial** function to do the actual dialing. Your **RASADFunc** then returns TRUE to indicate that it took over the dialing. When the dialing operation has been completed, set the variable pointed to by the *lpdwRetCode* parameter to indicate success or failure.

Your AutoDial DLL must provide both a **RASADFUNCA** (ANSI) and a **RASADFUNCW** (Unicode) version of the **RASADFunc** handler. To enable a **RASADFunc** AutoDial handler for a phone book entry, use the **RASENTRY** structure in a call to the **RasSetEntryProperties** function. The **szAutodialDll** member specifies the name of the DLL that contains the handler, and the **szAutodialFunc** member specifies the exported name of the handler. The **szAutodialFunc** member should not include the "A" or "W" suffix.

RASADFunc is a placeholder for the library-defined function name. The **RASADFUNC** type is a pointer to a **RASADFunc** function.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI prototypes.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasDial**, **RASENTRY**, **RasSetEntryProperties**

RasClearConnectionStatistics

The **RasClearConnectionStatistics** functions clears any accumulated statistics for the specified RAS connection.

```
DWORD RasClearConnectionStatistics (
  HRASCONN hRasConn    // handle to connection
);
```

Parameters

hRasConn

Handle to the connection. Use **RasDial** or **RasEnumConnections** to obtain this handle.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_INVALID_HANDLE	The *hRasConn* parameter does not specify a valid connection.
ERROR_NOT_ENOUGH_MEMORY	The function could not allocate sufficient memory to complete the operation.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasClearLinkStatistics**, **RasDial**, **RasEnumConnections**, **RasGetConnectionStatistics**, **RAS_STATS**

RasClearLinkStatistics

The **RasClearLinkStatistics** functions clears any accumulated statistics for the specified link in a RAS multilink connection.

```
DWORD RasClearLinkStatistics(
  HRASCONN hRasConn,    // handle to connection
  DWORD dwSubEntry      // SubEntry for link
);
```

Parameters

hRasConn

Handle to the connection. Use **RasDial** or **RasEnumConnections** to obtain this handle.

dwSubEntry

Specifies the subentry that corresponds to the link for which to clear statistics.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_INVALID_HANDLE	The *hRasConn* parameter does not specify a valid connection.
ERROR_INVALID_PARAMETER	The *dwSubEntry* parameter is zero.
ERROR_NO_CONNECTION	RAS could not find a connected port that corresponds to the value in the *dwSubEntry* parameter.
ERROR_NOT_ENOUGH_MEMORY	The function could not allocate sufficient memory to complete the operation.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasClearConnectionStatistics**, **RasGetLinkStatistics**

RasConnectionNotification

The **RasConnectionNotification** function specifies an event object that the system sets to the signaled state when a RAS connection is created or terminated.

```
DWORD RasConnectionNotification(
  HRASCONN hrasconn,  // handle to a RAS connection
  HANDLE hEvent,      // handle to an event object
  DWORD dwFlags       // type of event to receive notifications for
);
```

Parameters

hrasconn

Handle to the RAS connection for which to receive notifications. This can be a handle returned by the **RasDial** or **RasEnumConnections** function. If this parameter is INVALID_HANDLE_VALUE, you receive notifications for all RAS connections on the local computer.

hEvent

Specifies the handle of an event object. Use the **CreateEvent** function to create an event object.

dwFlags

Specifies the RAS event that causes the system to signal the event object specified by the *hEvent* parameter. This parameter can be a combination of the following values.

Value	Meaning
RASCN_Connection	If *hrasconn* is INVALID_HANDLE_VALUE, *hEvent* is signaled when any RAS connection is created.
RASCN_Disconnection	*hEvent* is signaled when the *hrasconn* connection is terminated. If *hrasconn* is a multilink connection, the event is signaled when all subentries are disconnected. If *hrasconn* is INVALID_HANDLE_VALUE, the event is signaled when any RAS connection is terminated.
RASCN_BandwidthAdded	**Windows NT 4.0 and earlier versions only:** If *hrasconn* is a handle to a combined multilink connection, *hEvent* is signaled when a subentry is connected.
RASCN_BandwidthRemoved	**Windows NT 4.0 and earlier versions only:** If *hrasconn* is a handle to a combined multilink connection, *hEvent* is signaled when a subentry is disconnected.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is a nonzero error code.

Remarks

To determine when the event object is signaled, use any of the wait functions.

When the event is signaled, you can use other RAS functions, such as **RasEnumConnections**, to get more information about the RAS connection that was created or terminated.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 98.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **CreateEvent**, **RasEnumConnections**

RasCreatePhonebookEntry

The **RasCreatePhonebookEntry** function creates a new phone book entry. The function displays a dialog box in which the user types information for the phone book entry.

Windows NT/2000: The **RasCreatePhonebookEntry** function calls the **RasEntryDlg** function. Applications written for Windows NT version 4.0 should use **RasEntryDlg**.

```
DWORD RasCreatePhonebookEntry(
  HWND hwnd,              // handle to the parent window
                          // of the dialog box
  LPCTSTR lpszPhonebook,  // pointer to the full path and
                          // file name of the phone book file
);
```

Parameters

hwnd
Handle to the parent window of the dialog box.

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

Windows 95: Dial-up networking stores phone book entries in the registry rather than in a phone book file.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is the following error code.

Value	Description
ERROR_CANNOT_OPEN_PHONEBOOK	The phone book is corrupted or missing components.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasEditPhonebookEntry**, **RasEntryDlg**, **RasGetEntryDialParams**, **RasSetEntryDialParams**

RasCustomDeleteEntryNotify

The **RasCustomDeleteEntryNotify** function is an application-defined function that is exported by a third-party custom-dialing DLL. This function allows third-party vendors to implement custom dialogs for managing phone book entries.

```
typedef DWORD (WINAPI *RasCustomDeleteEntryNotifyFn) (
  LPCTSTR lpszPhonebook,
  LPCTSTR lpszEntry,
  DWORD dwFlags
);
```

Parameters

lpszPhonebook
Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry
Pointer to a null-terminated string that contains the name of the phone book entry to dial.

dwFlags
Specifies one or more of the following flags:

RCD_SingleUser
RCD_AllUsers
RCD_Eap

Return Values

The function should return NO_ERROR.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCustomDial**, **RasCustomDialDlg**, **RasCustomEntryDlg**, **RasCustomHangUp**, **RasDial**,

RasCustomDial

The **RasCustomDial** function is an application-defined function that is exported by a third-party custom-dialing DLL. This function allows third-party vendors to implement custom remote-access dialing routines.

```
DWORD (WINAPI  * RasCustomDial) (
  HINSTANCE hInstDll,              // handle to DLL instance
  LPRASDIALEXTENSIONS lpRasDialExtensions,
                                  // pointer to function
                                   //extensions data
  LPCTSTR lpszPhonebook,          // pointer to full path
```

```
                                    //  and file name of
                                    //  phone book file
  LPRASDIALPARAMS lpRasDialParams,  // pointer to calling
                                    //  parameters data
  DWORD dwNotifierType,             // specifies type of
                                    //  RasDial event handler
  LPVOID lpvNotifier,               // specifies a handler
                                    //  for RasDial events
  LPHRASCONN lphRasConn             // pointer to variable
                                    //  to receive connection
                                    //handle
);
```

Parameters

hInstDll

Handle to the instance of the custom-dial DLL that was loaded.

lpRasDialExtensions

Pointer to a **RASDIALEXTENSIONS** structure that specifies a set of **RasDial** extended features to enable. If you do not need to enable any of the extensions, set this parameter to NULL.

lpszPhonebook

Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpRasDialParams

Pointer to a **RASDIALPARAMS** structure that specifies calling parameters for the RAS connection.

The caller must set the **RASDIALPARAMS** structure's **dwSize** member to **sizeof(RASDIALPARAMS)** to identify the version of the structure being passed.

dwNotifierType

This parameter is the same as the *dwNotifierType* parameter for the **RasDial** function. See the **RasDial** reference page for more information.

lpvNotifier

This parameter is the same as the *lpvNotifier* parameter for the **RasDial** function. See the **RasDial** reference page for more information.

lphRasConn

Pointer to a variable of type **HRASCONN**. You must set the **HRASCONN** variable to NULL before calling **RasDial**. If **RasDial** succeeds, it stores a handle to the RAS connection into **lphRasConn*.

Return Values

If the function succeeds, the immediate return value should be zero. In addition, the function should store a handle to the RAS connection into the variable pointed to by the *lphRasConn* parameter.

If the function fails, the immediate return value should be a nonzero error value, either from the set listed in Raserror.h or ERROR_NOT_ENOUGH_MEMORY.

Remarks

RAS calls this entry point from **RasDial**, if the **szCustomDialDll** member of the **RASENTRY** structure for the entry being dialed specifies a custom-dialing DLL.

If this entry point calls **RasDial**, the *lpRasDialExtensions* parameter must not be NULL, and the **dwFlags** member of the **RASDIALEXTENSIONS** structure must have the RDEOPT_CustomDial flag set.

If the custom-dial DLL does not support this entry point, RAS returns ERROR_CANNOT_DO_CUSTOMDIAL to the caller of **RasDial**.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI prototypes.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCustomDialDlg**, **RasCustomEntryDlg**, **RasCustomHangUp**, **RasDial**, **RASENTRY**

RasCustomDialDlg

The **RasCustomDialDlg** function is an application-defined function that is exported by a third-party custom-dialing DLL. This function allows third-party vendors to implement custom RAS connection dialog boxes.

```
BOOL (WINAPI  * RasCustomDialDlg) (
  HINSTANCE hInstDll,      // handle to DLL instance
  DWORD dwFlags,           // reserved
  LPTSTR lpszPhonebook,    // pointer to the full path and
                           // file name of the phone book
                           // file
  LPTSTR lpszEntry,        // pointer to the name of the
```

```
                          // phone book entry to dial
  LPTSTR lpszPhoneNumber, // pointer to replacement phone
                          // number to dial
  LPRASDIALDLG lpInfo     // pointer to a structure that
                          // contains additional
                          //parameters
);
```

Parameters

hInstDll
Handle to the instance of the custom-dialing DLL that was loaded.

dwFlags
The parameter is reserved for future use.

lpszPhonebook
Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry
Pointer to a null-terminated string that contains the name of the phone book entry to dial.

lpszPhoneNumber
Pointer to a null-terminated string that contains a phone number that overrides the numbers stored in the phone book entry. If this parameter is NULL, **RasDialDlg** uses the numbers in the phone book entry.

lpInfo
Pointer to a **RASDIALDLG** structure that contains additional input and output parameters. On input, the **dwSize** member of this structure must specify **sizeof(RASDIALDLG)**. If an error occurs, the **dwError** member returns an error code; otherwise, it returns zero.

Return Values

If the function establishes a RAS connection, the return value should be a nonzero value.

If an error occurs, or if the user selects a **Cancel** button during dialing box operation, the return value should be zero. If an error occurs, set the **dwError** member of the **RASDIALDLG** structure to a nonzero system error or a RAS error code from Raserror.h.

Remarks

RAS will call this entry point from **RasDialDlg**, if the **szCustomDialDll** member of the **RASENTRY** structure for the entry being dialed specifies a custom-dialing DLL.

If this entry point calls **RasDial**, the *lpRasDialExtensions* parameter must not be NULL, and the **dwFlags** member of the **RASDIALEXTENSIONS** structure must have the RDEOPT_CustomDial flag set.

The custom-dial dialog must support **WM_COMMAND** messages where **LOWORD**(*wParam*) equals IDCANCEL.

If the custom-dial DLL does not support this entry point, RAS returns ERROR_CANNOT_DO_CUSTOMDIAL to the caller of **RasDialDlg**.

! Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Unicode: Declared as Unicode and ANSI prototypes.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCustomDial**, **RasCustomEntryDlg**, **RasCustomHangUp**, **RasDialDlg**, **RASENTRY**

RasCustomEntryDlg

The **RasCustomEntryDlg** function is an application-defined function that is exported by a third-party custom-dialing DLL. This function allows third-party vendors to implement custom dialogs for managing phone book entries.

```
BOOL (WINAPI  * RasCustomEntryDlg) (
  HINSTANCE hInstDll,      // handle to DLL instance
  LPTSTR lpszPhonebook,    // pointer to the full path and
                           //  file name of the phone book
                           // file
  LPTSTR lpszEntry,        // pointer to the name of the
                           //  phone book entry to edit,
                           //  copy, or create
  LPRASENTRYDLG lpInfo     // pointer to a structure that
                           //  contains additional parameters
);
```

Parameters

hInstDll
Handle to the instance of the custom-dial DLL that was loaded.

lpszPhonebook

Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry

Pointer to a null-terminated string that contains the name of the phone book entry to edit, copy, or create.

If you are editing or copying an entry, this parameter is the name of an existing phone book entry. If you are copying an entry, set the RASEDFLAG_CloneEntry flag in the **dwFlags** member of the **RASENTRYDLG** structure.

If you are creating an entry, this parameter is a default new entry name that the user can change. If this parameter is NULL, the function provides a default name. If you are creating an entry, set the RASEDFLAG_NewEntry flag in the **dwFlags** member of the **RASENTRYDLG** structure.

lpInfo

Pointer to a **RASENTRYDLG** structure that contains additional input and output parameters. On input, the **dwSize** member of this structure must specify **sizeof**(**RASENTRYDLG**). Use the **dwFlags** member to indicate whether you are creating, editing, or copying an entry. If an error occurs, the **dwError** member returns an error code; otherwise, it returns zero.

Return Values

If the user creates, copies, or edits a phone book entry, the return value should be a nonzero value.

If an error occurs, or if the user cancels the operation, the return value should be zero. If an error occurs, the **RasCustomEntryDlg** should set the **dwError** member of the **RASENTRYDLG** structure to a nonzero system error code or a RAS error code from Raserror.h.

Remarks

RAS will call this entry point from **RasEntryDlg**, if the **szCustomDialDll** member of the **RASENTRY** structure for the entry being dialed specifies a custom-dialing DLL.

If the custom-dial DLL does not support this entry point, RAS returns ERROR_NO_CUSTOMENTRYDLG to the caller of **RasEntryDlg**.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Unicode: Declared as Unicode and ANSI prototypes.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCustomDial**, **RasCustomDialDlg**, **RasCustomHangUp**, **RasEntryDlg**, **RASENTRY**

RasCustomHangUp

The **RasCustomHangUp** function is an application-defined function that is exported by a third-party custom-dialing DLL. This function allows third-party vendors to implement custom connection hang-up routines.

```
DWORD (WINAPI  * RasCustomHangUp) (
  HRASCONN hRasConn    // handle to a RAS connection
);
```

Parameters

hRasConn
Handle to the RAS connection to hang up.

Return Values

If the function succeeds, the return value should be zero.

If the function fails, the return value should be a nonzero error value listed in Raserror.h, or ERROR_INVALID_HANDLE.

Remarks

RAS will call this entry point from **RasHangUp**, if the **szCustomDialDll** member of the **RASENTRY** structure for the entry being dialed specifies a custom-dialing DLL.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCustomDial**, **RasCustomDialDlg**, **RasCustomEntryDlg**, **RasHangUp**, **RASENTRY**

RasDeleteEntry

The **RasDeleteEntry** function deletes an entry from a phone book.

```
DWORD RasDeleteEntry(
  LPCTSTR lpszPhonebook,  // pointer to full path and file
                          //  name of phone book file
  LPCTSTR lpszEntry       // pointer to an entry name
                          //to delete
);
```

Parameters

lpszPhonebook
: **Windows NT/2000:** Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry
: Pointer to a null-terminated string containing the name of an existing entry to be deleted.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is ERROR_INVALID_NAME.

Remarks

The following sample code deletes the phone book entry specified by the variable *lpszEntry*:

```
nRet = RasDeleteEntry(NULL, lpszEntry);
if (nRet != ERROR_SUCCESS)
{
    printf("RasDeleteEntry failed: Error = %d\n", nRet);
}
else
{
    printf("Entry %s deleted successfully\n", lpszEntry);
}
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCreatePhonebookEntry**, **RasEnumEntries**

RasDial

The **RasDial** function establishes a RAS connection between a RAS client and a RAS server. The connection data includes callback and user-authentication information.

```
DWORD RasDial(
  LPRASDIALEXTENSIONS lpRasDialExtensions,
                          // pointer to function
                           // extensions data
  LPCTSTR lpszPhonebook,  // pointer to full path and file
                          // name of phone book file
  LPRASDIALPARAMS lpRasDialParams,
                          // pointer to calling
                           // parameters data
  DWORD dwNotifierType,   // specifies type of RasDial event
                           // handler
  LPVOID lpvNotifier,     // specifies a handler for
                           // RasDial events
  LPHRASCONN lphRasConn   // pointer to variable to receive
                          // connection handle
);
```

Parameters

lpRasDialExtensions

Windows NT/2000: Pointer to a **RASDIALEXTENSIONS** structure that specifies a set of **RasDial** extended features to enable. If you do not need to enable any of the extensions, set this parameter to NULL.

Windows 95: This parameter is ignored. On Windows 95, **RasDial** always uses the default behaviors for the **RASDIALEXTENSIONS** options.

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

Windows 95: This parameter is ignored. Dial-up networking stores phone book entries in the registry rather than in a phone book file.

lpRasDialParams

Pointer to a **RASDIALPARAMS** structure that specifies calling parameters for the RAS connection.

The caller must set the **RASDIALPARAMS** structure's **dwSize** member to **sizeof(RASDIALPARAMS)** to identify the version of the structure being passed.

dwNotifierType

Specifies the nature of the *lpvNotifier* parameter. If *lpvNotifier* is NULL, *dwNotifierType* is ignored. If *lpvNotifier* is not NULL, set *dwNotifierType* to one of the following values.

Value	Meaning
0xFFFFFFFF	The *lpvNotifier* parameter is a handle to a window to receive progress notification messages. In a progress notification message, *wParam* is the equivalent of the *rasconnstate* parameter of **RasDialFunc** and **RasDialFunc1**, and *lParam* is the equivalent of the *dwError* parameter of **RasDialFunc** and **RasDialFunc1**. The progress notification message uses a system registered message code. You can obtain the value of this message code as follows: `{UINT unMsg = RegisterWindowMessageA( RASDIALEVENT ); if (unMsg == 0) unMsg = WM_RASDIALEVENT; }`
0	The *lpvNotifier* parameter points to a **RasDialFunc** callback function.
1	The *lpvNotifier* parameter points to a **RasDialFunc1** callback function.
2	**Windows NT/2000:** The *lpvNotifier* parameter points to a **RasDialFunc2** callback function.

lpvNotifier

Specifies a window handle or a **RasDialFunc**, **RasDialFunc1**, or **RasDialFunc2** callback function to receive **RasDial** event notifications. The *dwNotifierType* parameter specifies the nature of *lpvNotifier*. Please refer to its description preceding for further detail.

If this parameter is not NULL, **RasDial** sends the window a message, or calls the callback function, for each **RasDial** event. Additionally, the **RasDial** call operates asynchronously: **RasDial** returns immediately, before the connection is established, and communicates its progress via the window or callback function.

If *lpvNotifier* is NULL, the **RasDial** call operates synchronously: **RasDial** does not return until the connection attempt has completed successfully or failed.

If *lpvNotifier* is not NULL, notifications to the window or callback function can occur at any time after the initial call to **RasDial**. Notifications end when one of the following events occurs.

- The connection is established. In other words, the RAS connection state is RASCS_Connected.
- The connection fails. In other words, *dwError* is nonzero.
- **RasHangUp** is called on the connection.

The callback notifications are made in the context of a thread captured during the initial call to **RasDial**.

lphRasConn

Pointer to a variable of type **HRASCONN**. You must set the **HRASCONN** variable to NULL before calling **RasDial**. If **RasDial** succeeds, it stores a handle to the RAS connection into **lphRasConn*.

Return Values

If the function succeeds, the immediate return value is zero. In addition, the function stores a handle to the RAS connection into the variable pointed to by *lphRasConn*.

If the function fails, the immediate return value is a nonzero error value, either from the set listed in the RAS header file or ERROR_NOT_ENOUGH_MEMORY.

Remarks

Errors that occur after the immediate return can be detected by **RasGetConnectStatus**. Data is available until an application calls **RasHangUp** to hang up the connection.

An application must eventually call **RasHangUp** whenever a non-NULL connection handle is stored into **lphRasConn*. This applies even if **RasDial** returns a nonzero (error) value.

An application can safely call **RasHangUp** from a **RasDial** notifier callback function. If this is done, however, the hang-up does not occur until the routine returns.

Windows NT/2000: If the structure pointed to by *lpRasDialExtensions* enables RDEOPT_PausedStates, the **RasDial** function pauses whenever it enters a state in which the RASCS_PAUSED bit is set to one. To restart **RasDial** from such a paused state, call **RasDial** again, passing the connection handle returned from the original **RasDial** call in **lphRasConn*. The same notifier used in the original **RasDial** call must be used when restarting from a paused state.

Windows 2000: RAS supports referenced connections. If the entry being dialed is already connected, **RasDial** will return SUCCESS and the connection will be referenced. To disconnect the connection, each **RasDial** on the connection should be matched by a **RasHangUp**.

Windows 2000: Because some phone book entries require Extensible Authentication Protocol (EAP) for authentication, the caller should call **RasGetEapUserIdentity** before calling **RasDial**. If **RasGetEapUserIdentity** returns ERROR_INVALID_ENTRY_FOR_FUNCTION, the phone book entry does not require EAP. However, if **RasGetEapUserIdentity** returns NO_ERROR, the caller should copy the EAP identity information from **RasGetEapUserIdentity** into the **RasEapInfo** member of **RASDIALEXTENSIONS**, and the **szUserName** member of **RASDIALPARAMS**. See **RasGetEapUserIdentity** for more information. If the phone book entry requires EAP, the **dwfOptions** member of the **RASENTRY** structure for the entry contains the RASEO_RequireEAP flag.

To specify that **RasDial** should enter a RASCS_CallbackSetByCaller state, set *lpRasDialParams*->**szCallbackNumber** to "*" on the initial call to **RasDial**. When your notification handler is called with this state, you can set the callback number to a number supplied by the user.

Windows 95: Windows 95 does not support the RASCS_CallbackSetByCaller state or any of the other paused states.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, Dialable Addresses, **RasDialDlg**, **RasDialFunc**, **RasDialFunc1**, **RasDialFunc2**, **RasGetConnectStatus**, **RasHangUp**, **RASDIALEXTENSIONS**, **RASDIALPARAMS**, WM_RASDIALEVENT

RasDialDlg

The **RasDialDlg** function establishes a RAS connection using a specified phone book entry and the credentials of the logged-on user. The function displays a stream of dialog boxes that indicate the state of the connection operation.

```
BOOL RasDialDlg(
  LPTSTR lpszPhonebook,    // pointer to the full path and
                           // file name of the phone book
                           // file
  LPTSTR lpszEntry,        // pointer to the name of the
                           //  phone book entry to dial
  LPTSTR lpszPhoneNumber,  // pointer to replacement phone
                           //  number to dial
  LPRASDIALDLG lpInfo      // pointer to a structure that
                           //  contains additional parameters
);
```

Parameters

lpszPhonebook
Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry
Pointer to a null-terminated string that contains the name of the phone book entry to dial.

lpszPhoneNumber
Pointer to a null-terminated string that contains a phone number that overrides the numbers stored in the phone book entry. If this parameter is NULL, **RasDialDlg** uses the numbers in the phone book entry.

lpInfo
Pointer to a **RASDIALDLG** structure that contains additional input and output parameters. On input, the **dwSize** member of this structure must specify **sizeof**(**RASDIALDLG**). If an error occurs, the **dwError** member returns an error code; otherwise, it returns zero.

Return Values

If the function establishes a RAS connection, the return value is a nonzero value.

If an error occurs, or if the user selects the **Cancel** button during the dialing operation, the return value is zero. If an error occurs, the **dwError** member of the **RASDIALDLG** structure returns a nonzero system or RAS error code.

Remarks

The **RasDialDlg** function displays a series of dialog boxes that are similar to the dialog boxes that main **Dial-Up Networking** dialog box displays when the user selects the **Dial** button. The **RasDialDlg** function is useful for applications in which you want to display a standard user interface for a connection operation without presenting the main phone book dialog box. For example, the RAS AutoDial service uses this function to establish a connection using the phone book entry associated with a remote address.

The **RasDialDlg** function displays dialog boxes during the connection operation to provide feedback to the user about the progress of the operation. For example, the dialog boxes might indicate when the operation is dialing, when it is authenticating the user's credentials on the remote server, and so on. The dialog boxes also provide a **Cancel** button for the user to terminate the operation.

RasDialDlg returns when the connection is established, or when the user cancels the operation.

The sample code on the following page dials the entry in the default phone book specified by the variable *lpszEntry*.

```
lpInfo = (LPRASDIALDLG) GlobalAlloc(GPTR, sizeof(RASDIALDLG));
lpInfo->dwSize = sizeof(RASDIALDLG);

printf("Dialing %s...\n", lpszEntry);

//    Calling RasDialDlg()

nRet = RasDialDlg(NULL, lpszEntry, NULL, lpInfo);
if (nRet == 0)
{
    printf("RasDialDlg failed: Error = %d\n", lpInfo->dwError);
}
else
{
    printf("Connection established.\n");
}
GlobalFree(lpInfo);
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Library: Use Rasdlg.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASDIALDLG**, **RasPhonebookDlg**

RasDialFunc

The **RasDialFunc** function is an application-defined or library-defined callback function that the **RasDial** function calls when a change of state occurs during a RAS connection process.

```
VOID WINAPI RasDialFunc(
  UINT unMsg,     // type of event that has occurred
  RASCONNSTATE rasconnstate,
                  // connection state about to be entered
  DWORD dwError   // error that may have occurred
);
```

Parameters

unMsg
Specifies the type of event that has occurred. Currently, the only event defined is WM_RASDIALEVENT.

rasconnstate
Specifies a **RASCONNSTATE** enumerator value that indicates the state the **RasDial** remote access connection process is about to enter.

dwError
Specifies the error that has occurred, or zero if no error has occurred.

RasDial calls **RasDialFunc** with *dwError* set to zero upon entry to each connection state. If an error occurs within a state, **RasDialFunc** is called again with a nonzero *dwError* value.

Return Values

None.

Remarks

A **RasDial** connection operation is suspended during a call to a **RasDialFunc** callback function. For that reason, your **RasDialFunc** implementation should generally return as quickly as possible. There are two exceptions to that rule. Asynchronous (slow) devices such as modems often have time-out periods measured in seconds rather than milliseconds; a slow return from a **RasDialFunc** function is generally not a problem. The prompt return requirement also does not apply when *dwError* is nonzero, indicating that an error has occurred. It is safe, for example, to put up an error dialog box and wait for user input.

Your **RasDialFunc** implementation should not depend on the order or occurrence of particular **RASCONNSTATE** connection states, because this may vary between platforms.

Do not call the **RasDial** function from within a **RasDialFunc** callback function. You can call the **RasGetConnectStatus**, **RasEnumEntries**, **RasEnumConnections**, **RasGetErrorString**, and **RasHangUp** functions from within the callback function. For example, calling **RasGetConnectStatus** from within a callback function would be useful for determining the name and type of the connecting device.

Note For convenience, **RasHangUp** can be called from within a **RasDialFunc** callback function. However, much of the hang-up processing occurs after the **RasDialFunc** callback function has returned.

RasDialFunc is a placeholder for the application-defined or library-defined function name.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASCONNSTATE**, **RasDial**, **RasDialFunc1**, **RasDialFunc2**, **RasEnumConnections**, **RasEnumEntries**, **RasGetConnectStatus**, **RasGetErrorString**, **RasHangUp**

RasDialFunc1

A **RasDialFunc1** function is an application-defined or library-defined callback function that the **RasDial** function calls when a change of state occurs during a remote access connection process. A **RasDialFunc1** function is comparable to a **RasDialFunc** function, but is enhanced by the addition of two parameters: a handle to the RAS connection, and an extended error code.

```
VOID WINAPI RasDialFunc1(
  HRASCONN hrasconn,   // handle to RAS connection
  UINT unMsg,          // type of event that has occurred
  RASCONNSTATE rascs,  // connection state about to
                       // be entered
  DWORD dwError,       // error that may have occurred
  DWORD dwExtendedError
                       // extended error information
                       // for some errors
);
```

Parameters

hrasconn
Handle to the RAS connection, as returned by **RasDial**.

unMsg
Specifies the type of event that has occurred. Currently, the only event defined is WM_RASDIALEVENT.

rascs
Specifies a **RASCONNSTATE** enumerator value that indicates the state the **RasDial** remote access connection process is about to enter.

dwError

Specifies the error that has occurred. If no error has occurred, *dwError* is zero.

RasDial calls **RasDialFunc1** with *dwError* set to zero upon entry to each connection state. If an error occurs within a state, **RasDial** calls **RasDialFunc1** again with a nonzero *dwError* value.

In some error cases, the *dwExtendedError* parameter contains extended error information.

dwExtendedError

Specifies extended error information for certain nonzero values of *dwError*. For all other values of *dwError, dwExtendedError* is zero.

The contents of *dwExtendedError* are defined for values of *dwError* as follows.

dwError	***dwExtendedError***
ERROR_SERVER_NOT_RESPONDING	Specifies the NetBIOS error that occurred.
ERROR_NETBIOS_ERROR	Specifies the NetBIOS error that occurred.
ERROR_AUTH_INTERNAL	Specifies an internal diagnostics code.
ERROR_CANNOT_GET_LANA	Specifies a routing error code, which is a RAS error.

Return Values

None.

Remarks

A **RasDial** connection operation is suspended during a call to a **RasDialFunc1** callback function. For that reason, your **RasDialFunc1** implementation should generally return as quickly as possible. There are two exceptions to that rule. Asynchronous (slow) devices such as modems often have time-out periods measured in seconds rather than milliseconds; a slow return from a **RasDialFunc1** function is generally not a problem. The prompt return requirement also does not apply when *dwError* is nonzero, indicating that an error has occurred. It is safe, for example, to put up an error dialog box and wait for user input.

Your **RasDialFunc1** implementation should not depend on the order or occurrence of particular **RASCONNSTATE** connection states, because this may vary between platforms.

Do not call the **RasDial** function from within a **RasDialFunc1** callback function. You can call the **RasGetConnectStatus**, **RasEnumEntries**, **RasEnumConnections**, **RasGetErrorString**, and **RasHangUp** functions from within the callback function. For example, calling **RasGetConnectStatus** from within a callback function would be useful for determining the name and type of the connecting device.

Note that, for convenience, **RasHangUp** can be called from within a **RasDialFunc1** callback function. However, much of the hang-up processing occurs after the **RasDialFunc1** callback function has returned.

RasDialFunc1 is a placeholder for the application-defined or library-defined function name.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasDial**, **RasDialFunc**, **RasDialFunc2**, **RASCONNSTATE**, **RasEnumConnections**, **RasEnumEntries**, **RasGetConnectStatus**, **RasGetErrorString**, **RasHangUp**

RasDialFunc2

A **RasDialFunc2** function is an application-defined or library-defined callback function that the **RasDial** function calls when a change of state occurs during a remote access connection process. A **RasDialFunc2** function is similar to the **RasDialFunc1** callback function, except that it provides additional information for multilink connections.

```
DWORD WINAPI RasDialFunc2(
  DWORD dwCallbackId,     // user-defined value specified in
                          //  RasDial call
  DWORD dwSubEntry,       // subentry index in multilink
                          // connection
  HRASCONN hrasconn,      // handle to RAS connection
  UINT unMsg,             // type of event that has occurred
  RASCONNSTATE rascs,     // connection state about to
                          //  be entered
  DWORD dwError,          // error that may have occurred
  DWORD dwExtendedError   // extended error information for
                          //  some errors
);
```

Parameters

dwCallbackId

Provides an application-defined value that was specified in the **dwCallbackId** member of the **RASDIALPARAMS** structure passed to **RasDial**.

dwSubEntry
Specifies a subentry index for the phone book entry associated with this connection. This value indicates the subentry that generated this call to your **RasDialFunc2** callback function.

hrasconn
Handle to the RAS connection, as returned by **RasDial**.

unMsg
Specifies the type of event that has occurred. Currently, the only event defined is WM_RASDIALEVENT.

rascs
Specifies a **RASCONNSTATE** enumerator value that indicates the state the **RasDial** remote access connection process is about to enter.

dwError
Specifies the error that has occurred. If no error has occurred, *dwError* is zero.

The **RasDial** function calls **RasDialFunc2** with *dwError* set to zero upon entry to each connection state. If an error occurs within a state, **RasDial** calls **RasDialFunc2** again with a nonzero *dwError* value.

In some error cases, the *dwExtendedError* parameter contains extended error information.

dwExtendedError
Specifies extended error information for certain nonzero values of *dwError*. For all other values of *dwError, dwExtendedError* is zero.

The contents of *dwExtendedError* are defined for values of *dwError* as follows.

dwError	***dwExtendedError***
ERROR_SERVER_NOT_RESPONDING	Specifies the NetBIOS error that occurred.
ERROR_NETBIOS_ERROR	Specifies the NetBIOS error that occurred.
ERROR_AUTH_INTERNAL	Specifies an internal diagnostics code.
ERROR_CANNOT_GET_LANA	Specifies a routing error code, which is a RAS error.

Return Values

If the **RasDialFunc2** function returns a nonzero value, **RasDial** continues to send callback notifications.

If the **RasDialFunc2** function returns zero, **RasDial** stops sending callback notifications for all subentries.

Remarks

A **RasDial** connection operation is suspended during a call to a **RasDialFunc2** callback function. For that reason, your **RasDialFunc2** implementation should generally return as quickly as possible. There are two exceptions to that rule. Asynchronous (slow) devices such as modems often have time-out periods measured in seconds rather than milliseconds; a slow return from a **RasDialFunc2** function is generally not a problem. The prompt return requirement also does not apply when *dwError* is nonzero, indicating that an error has occurred. It is safe, for example, to put up an error dialog box and wait for user input.

Your **RasDialFunc2** implementation should not depend on the order or occurrence of particular **RASCONNSTATE** connection states, because this may vary between platforms.

Do not call the **RasDial** function from within a **RasDialFunc2** callback function. You can call the **RasGetConnectStatus**, **RasEnumEntries**, **RasEnumConnections**, **RasGetErrorString**, and **RasHangUp** functions from within the callback function. For example, calling **RasGetConnectStatus** from within a callback function would be useful for determining the name and type of the connecting device.

Note For convenience, **RasHangUp** can be called from within a **RasDialFunc2** callback function. However, much of the hang-up processing occurs after the **RasDialFunc2** callback function has returned.

RasDialFunc2 is a placeholder for the application-defined or library-defined function name.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasDial**, **RasDialFunc**, **RasDialFunc1**, **RASCONNSTATE**, **RasEnumConnections**, **RasEnumEntries**, **RasGetConnectStatus**, **RasGetErrorString**, **RasHangUp**

RasEditPhonebookEntry

The **RasEditPhonebookEntry** function edits an existing phone book entry. The function displays a dialog box in which the user can modify the existing information.

Windows NT/2000: The **RasEditPhonebookEntry** function calls the **RasEntryDlg** function. Applications written for Windows NT version 4.0 should use **RasEntryDlg**.

```
DWORD RasEditPhonebookEntry(
  HWND hwnd,              // handle to the parent window of
                          //  the dialog box
  LPCTSTR lpszPhonebook,  // pointer to the full path and
                          //  file name of the phone book
                          // file
  LPCTSTR lpszEntryName   // pointer to the phone book
                          // entry name
);
```

Parameters

hwnd

Handle to the parent window of the dialog box.

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

Windows 95: Dial-up networking stores phone book entries in the registry rather than in a phone book file.

lpszEntryName

Pointer to a null-terminated string that specifies the name of an existing entry in the phone book file.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Description
ERROR_BUFFER_INVALID	The phone book entry buffer is invalid.
ERROR_CANNOT_OPEN_PHONEBOOK	The phone book is corrupted or missing components.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	The phone book entry does not exist.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCreatePhonebookEntry**, **RasEntryDlg**, **RasGetEntryDialParams**, **RasSetEntryDialParams**

RasEntryDlg

The **RasEntryDlg** function displays modal property sheets that allow a user to manipulate phone book entries. If editing or copying an existing phone book entry, the function displays a phone book entry property sheet. The **RasEntryDlg** function returns when the user closes the property sheet.

```
BOOL RasEntryDlg(
  LPTSTR lpszPhonebook, // pointer to the full path and
                        // file name of the phone book file
  LPTSTR lpszEntry,     // pointer to the name of the phone-
                        // book entry to edit, copy, or
                        // create
  LPRASENTRYDLG lpInfo  // pointer to a structure that
                        //  contains additional parameters
);
```

Parameters

lpszPhonebook

Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry

Pointer to a null-terminated string that contains the name of the phone book entry to edit, copy, or create.

If you are editing or copying an entry, this parameter is the name of an existing phone book entry. If you are copying an entry, set the RASEDFLAG_CloneEntry flag in the **dwFlags** member of the **RASENTRYDLG** structure.

If you are creating an entry, this parameter is a default new entry name that the user can change. If this parameter is NULL, the function provides a default name. If you are creating an entry, set the RASEDFLAG_NewEntry flag in the **dwFlags** member of the **RASENTRYDLG** structure.

lpInfo

Pointer to a **RASENTRYDLG** structure that contains additional input and output parameters. On input, the **dwSize** member of this structure must specify **sizeof**(**RASENTRYDLG**). Use the **dwFlags** member to indicate whether you are creating, editing, or copying an entry. If an error occurs, the **dwError** member returns an error code; otherwise, it returns zero.

Return Values

If the user creates, copies, or edits a phone book entry, the return value is a nonzero value.

If an error occurs, or if the user cancels the operation, the return value is zero. If an error occurs, the **dwError** member of the **RASENTRYDLG** structure returns a nonzero system error code or RAS error code.

Remarks

The **RasCreatePhonebookEntry** and **RasEditPhonebookEntry** functions call the **RasEntryDlg** function.

The following sample code brings up a property sheet to create a new entry. The lpEntry variable specifies the default name for the new entry.

```
lpInfo = (LPRASENTRYDLG) GlobalAlloc(GPTR, sizeof(RASENTRYDLG));
ZeroMemory(lpInfo, sizeof(RASENTRYDLG));
lpInfo->dwSize = sizeof(RASENTRYDLG);

lpInfo->dwFlags |= RASEDFLAG_NewEntry;
nRet = RasEntryDlg(NULL, lpszEntry, lpInfo);

if (nRet)
    printf("New entry created: %s\n", lpInfo->szEntry);
else
{
    if (lpInfo->dwError != 0)
    {
        printf("RasEntryDlg failed: Error = %d\n", lpInfo->dwError);
    }
    else
        printf("User pressed Cancel\n");
}
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Library: Use Rasdlg.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCreatePhonebookEntry**, **RasEditPhonebookEntry**, **RASENTRYDLG**

RasEnumAutodialAddresses

The **RasEnumAutodialAddresses** function returns a list of all addresses in the AutoDial mapping database.

```
DWORD RasEnumAutodialAddresses(
  LPTSTR *lppAddresses,     // pointer to buffer that
                            // receives network address
                            // strings
  LPDWORD lpdwcbAddresses,  // pointer to size, in bytes,
                            //  of the buffer
  LPDWORD lpdwcAddresses    // pointer to number of strings
                            //  returned
);
```

Parameters

lppAddresses
Pointer to an array of string pointers, with additional space for the storage of the strings themselves at the end of the buffer. Each string is the name of an address in the AutoDial mapping database.

If *lppAddresses* is NULL, **RasEnumAutodialAddresses** sets the *lpdwcbAddresses* and *lpdwcAddresses* parameters to indicate the required size, in bytes, and the number of address entries in the database.

lpdwcbAddresses
Pointer to a variable that contains the size, in bytes, of the buffer specified by the *lppAddresses* parameter. On return, the function sets this variable to the number of bytes returned, or the number of bytes required if the buffer is too small.

lpdwcAddresses
Pointer to a variable that receives the number of address strings returned in the *lppAddresses* buffer.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is the following error code.

Value	Meaning
ERROR_INVALID_PARAMETER	NULL was passed for the *lpdwcbAddresses* or *lpdwcAddresses* parameter.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASAUTODIALENTRY**, **RasGetAutodialAddress**, **RasSetAutodialAddress**

RasEnumConnections

The **RasEnumConnections** function lists all active RAS connections. It returns each connection's handle and phone book entry name.

```
DWORD RasEnumConnections(
  LPRASCONN lprasconn,      // buffer to receive connections
                            //  data
  LPDWORD lpcb,             // size in bytes of buffer
  LPDWORD lpcConnections    // number of connections written
                            //  to buffer
);
```

Parameters

lprasconn
Pointer to a buffer that receives an array of **RASCONN** structures, one for each RAS connection. Before calling the function, an application must set the **dwSize** member of the first **RASCONN** structure in the buffer to **sizeof(RASCONN)** in order to identify the version of the structure being passed.

lpcb
Pointer to a variable that contains the size, in bytes, of the buffer specified by *lprasconn*. On return, the function sets this variable to the number of bytes required to enumerate the RAS connections.

lpcConnections
Pointer to a variable that the function sets to the number of **RASCONN** structures written to the buffer specified by *lprasconn*.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is either a nonzero error value listed in the RAS header file or ERROR_BUFFER_TOO_SMALL or ERROR_NOT_ENOUGH_MEMORY.

Remarks

If a connection was made without specifying a phone book entry name, the information returned for that connection gives the connection phone number preceded by ".".

The following sample code enumerates the current RAS connection. This code assumes that, at most, only one connection is currently active. Note that the code sets the **dwSize** member of the **RASCONN** structure to specify the version of the structure:

```
lpRasConn = (LPRASCONN) GlobalAlloc(GPTR, sizeof(RASCONN));
lpRasConn->dwSize = sizeof(RASCONN);

nRet = RasEnumConnections(lpRasConn, &lpcb, &lpcConnections);
if (nRet != 0)
{
    printf("RasEnumConnections failed: Error = %d", nRet);
}
else
{
    printf("The following RAS connections are currently active\n\n");
    for (i = 0; i < lpcConnections; i++)
    {
        printf("Entry name: %s\n", lpRasConn->szEntryName);
        lpRasConn++;
    }
}
```

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASCONN**, **RasEnumEntries**, **RasGetConnectStatus**

RasEnumDevices

The **RasEnumDevices** function returns the name and type of all available RAS-capable devices.

```
DWORD RasEnumDevices(
  LPRASDEVINFO lpRasDevInfo,  // buffer to receive
                              // information about
                              // RAS devices
  LPDWORD lpcb,               // size, in bytes, of
                              // the buffer
  LPDWORD lpcDevices          // receives the number of
                              //  entries written to the buffer
);
```

Parameters

lpRasDevInfo

Pointer to a buffer that receives an array of **RASDEVINFO** structures, one for each RAS-capable device. Before calling the function, set the **dwSize** member of the first **RASDEVINFO** structure in the buffer to `sizeof(RASDEVINFO)` to identify the version of the structure.

lpcb

Pointer to a variable that contains the size, in bytes, of the *lpRasDevInfo* buffer. On return, the function sets this variable to the number of bytes required to enumerate the devices.

To determine the required buffer size, call **RasEnumDevices** with the *lpRasDevInfo* parameter set to NULL and the variable pointed to by *lpcb* set to zero. The function returns the required buffer size in the variable pointed to by *lpcb*. (See sample code under **Remarks** section.)

lpcDevices

Pointer to a variable that the function sets to the number of **RASDEVINFO** structures written to the *lpRasDevInfo* buffer.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is a nonzero RAS error value or one of following error codes.

Value	Meaning
ERROR_BUFFER_TOO_SMALL	The *lpRasDevInfo* buffer is not large enough. The function returns the required buffer size in the variable pointed to by *lpcb*.
ERROR_NOT_ENOUGH_MEMORY	Indicates insufficient memory.
ERROR_INVALID_PARAMETER	Indicates an invalid parameter value.
ERROR_INVALID_USER_BUFFER	The address or buffer specified by *lpRasDevInfo* is invalid.

Remarks

The following sample code enumerates the devices on the current machine. The code initially calls **RasEnumDevices** with a *lpRasDevInfo* parameter of NULL, to obtain the size of the buffer that should be passed in. The code also sets the **dwSize** member of the first **RASDEVINFO** structure to sizeof(**RASDEVINFO**) to specify the version of the structure.

```
RasEnumDevices(NULL, &lpcb, &lpcDevices);
lpRasDevInfo = (LPRASDEVINFO) GlobalAlloc(GPTR, lpcb);
lpRasDevInfo->dwSize = sizeof(RASDEVINFO);

nRet = RasEnumDevices(lpRasDevInfo, &lpcb, &lpcDevices);
if (nRet != 0)
{
    printf("RasEnumDevices failed: Error %d", nRet);
}
else
{
    printf("The following RAS capable devices were found on this machine:\n\n");
    for (i=0; i < lpcDevices; i++)
    {
        printf("%s\n",lpRasDevInfo->szDeviceName);
        lpRasDevInfo++;
    }
}
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASDEVINFO**

RasEnumEntries

The **RasEnumEntries** function lists all entry names in a remote access phone book.

```
DWORD RasEnumEntries (
  LPCTSTR reserved,              // reserved, must be NULL
```

(continued)

(continued)

```
  LPTCSTR lpszPhonebook,         // pointer to full path and
                                  // file name of phone-
                                  // book file
  LPRASENTRYNAME lprasentryname, // buffer to receive
                                  //  phone book entries
  LPDWORD lpcb,                  // size in bytes of buffer
  LPDWORD lpcEntries             // number of entries
                                 // written to buffer
);
```

Parameters

reserved

Reserved; must be NULL.

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

Windows 2000: If this parameter is NULL, the entries are enumerated from all the remote access phone book files in the AllUsers profile and the user's profile.

Windows 95: This parameter is ignored. Dial-up networking stores phone book entries in the registry rather than in a phone book file.

lprasentryname

Pointer to a buffer that receives an array of **RASENTRYNAME** structures, one for each phone book entry. Before calling the function, an application must set the **dwSize** member of the first **RASENTRYNAME** structure in the buffer to **sizeof(RASENTRYNAME)** in order to identify the version of the structure being passed.

lpcb

Pointer to a variable that contains the size, in bytes, of the buffer specified by *lprasentryname*. On return, the function sets this variable to the number of bytes required to successfully complete the call.

lpcEntries

Pointer to a variable that the function, if successful, sets to the number of phone book entries written to the buffer specified by *lprasentryname*.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is a nonzero error value listed in the RAS header file or one of the following values.

Value	Meaning
ERROR_BUFFER_TOO_SMALL	The buffer pointed to by the *lprasentryname* parameter is not large enough to hold all the entries.
ERROR_INVALID_SIZE	The value of **dwSize** in the **RASENTRYNAME** structure pointed to by *lprasentryname*, specifies a version of the structure that is not supported on the current platform. For example, on Windows 95, **RasEnumEntries** returns this error if **dwSize** indicates that **RASENTRYNAME** includes the **dwFlags** and **szPhonebookPath** members, since these members are not supported on Windows 95 (they are supported only on Windows 2000 and later).
ERROR_NOT_ENOUGH_MEMORY	The function could not allocate sufficient memory to complete the operation.

Remarks

The following sample code enumerates the RAS phone book entries on the current machine. The code initially calls **RasEnumEntries** to obtain the size of the buffer to pass in. The code then calls **RasEnumEntries** again, to enumerate the entries. Note that for both calls, the code sets the **dwSize** member of the first **RASENTRY** structure in the buffer to sizeof(**RASENTRY**) to specify the structure version.

```
lpRasEntryName = (LPRASENTRYNAME)GlobalAlloc(GPTR, sizeof(RASENTRYNAME));
lpRasEntryName->dwSize = sizeof(RASENTRYNAME);
if ((nRet = RasEnumEntries(NULL, NULL, lpRasEntryName, &cb, &cEntries))
    == ERROR_BUFFER_TOO_SMALL)
{
    lpRasEntryName = (LPRASENTRYNAME)GlobalAlloc(GPTR, cb);
    lpRasEntryName->dwSize = sizeof(RASENTRYNAME);
}

// Calling RasEnumEntries to enumerate the phone book entries
nRet = RasEnumEntries(NULL, NULL, lpRasEntryName, &cb, &cEntries);

if (nRet != ERROR_SUCCESS)
{
    printf("RasEnumEntries failed: Error %d\n", nRet);
}
else
{
    printf("Phone book entries in the default phone book:\n\n");
    for(i=0;i < cEntries;i++)
    {
        printf("%s\n",lpRasEntryName->szEntryName);
```

(continued)

(continued)

```
        lpRasEntryName++;
    }
}
```

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASENTRYNAME**, **RasEnumConnections**

RasFreeEapUserIdentity

Use the **RasFreeEapUserIdentity** function to free the memory buffer returned by **RasGetEapUserIdentity**.

```
DWORD RasFreeEapUserIdentity(
  LPRASEAPUSERIDENTITY pRasEapUserIdentity
                              // pointer to memory to free
);
```

Parameters

pRasEapUserIdentity
Pointer to the **RASEAPUSERIDENTITY** structure returned by the **RasGetEapUserIdenity** function.

Return Values

If the function succeeds, the return value is NO_ERROR.

Otherwise, the function returns one of the following error codes.

Remarks

RasFreeEapUserIdentity may be called with the *pRasEapUserIdentity* parameter equal to NULL.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows 2000.

See Also

RASEAPUSERIDENTITY, **RasGetEapUserIdentity**

RasGetAutodialAddress

The **RasGetAutodialAddress** function retrieves information about all the AutoDial entries associated with a network address in the AutoDial mapping database.

```
DWORD RasGetAutodialAddress(
  LPCTSTR lpszAddress,    // pointer to a network
                          // address string
  LPDWORD lpdwReserved,   // reserved; must be NULL
  LPRASAUTODIALENTRY lpAutoDialEntries,
                          // pointer to buffer for
                           // AutoDial entry data
  LPDWORD lpdwcbAutoDialEntries,
                          // pointer to size, in bytes,
                           // of buffer
  LPDWORD lpdwcAutoDialEntries
                          // pointer to number of
                          // entries returned
);
```

Parameters

lpszAddress
: Pointer to a null-terminated string that specifies the address for which information is requested. This can be an IP address, Internet host name ("www.microsoft.com"), or NetBIOS name ("products1").

lpdwReserved
: Reserved; must be NULL.

lpAutoDialEntries
: Pointer to a buffer that receives an array of **RASAUTODIALENTRY** structures, one for each AutoDial entry associated with the address specified by the *lpszAddress* parameter. Before calling **RasGetAutodialAddress**, set the **dwSize** member of the first **RASAUTODIALENTRY** structure in the buffer to `sizeof(RASAUTODIALENTRY)` to identify the version of the structure.

If *lpAutoDialEntries* is NULL, **RasGetAutodialAddress** sets the *lpdwcbAutoDialEntries* and *lpdwcAutoDialEntries* parameters to indicate the required buffer size, in bytes, and the number of AutoDial entries.

lpdwcbAutoDialEntries
Pointer to a variable that contains the size, in bytes, of the *lpAutoDialEntries* buffer. On return, the function sets this variable to the number of bytes returned, or the number of bytes required if the buffer is too small.

lpdwcAutoDialEntries
Pointer to a variable that receives the number of structure elements returned in the *lpAutoDialEntries* buffer.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_XXX_NOT_FOUND	The address was not found in the mapping database.
ERROR_INVALID_SIZE	The **dwSize** member of the **RASAUTODIALENTRY** structure is an invalid value.
ERROR_INVALID_PARAMETER	The *lpszAddress*, *lpdwcbAutoDialEntries*, or *lpdwcAutoDialEntries* parameter was NULL.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASAUTODIALENTRY**, **RasEnumAutodialAddresses**, **RasSetAutodialAddress**

RasGetAutodialEnable

The **RasGetAutodialEnable** function indicates whether the AutoDial feature is enabled for a specified TAPI dialing location. For more information about TAPI dialing locations, see the *TAPI Programmer's Reference* in the *Platform SDK*.

```
DWORD RasGetAutodialEnable(
  DWORD dwDialingLocation,
                      // identifier of the TAPI
                      // dialing location
  LPBOOL lpfEnabled   // pointer to variable that receives
                      //  AutoDial state for this location
);
```

Parameters

dwDialingLocation
  Specifies the identifier of a TAPI dialing location.

lpfEnabled
  Pointer to a BOOL variable that receives a nonzero value if AutoDial is enabled for the specified dialing location, or zero if it is not enabled.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is a nonzero value.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasSetAutodialEnable**

RasGetAutodialParam

The **RasGetAutodialParam** function retrieves the value of an AutoDial parameter.

```
DWORD RasGetAutodialParam(
  DWORD dwKey,       // indicates the parameter to retrieve
  LPVOID lpvValue,   // pointer to a buffer that receives
                     // the value
  LPDWORD lpdwcbValue
                     // size, in bytes, of the buffer
);
```

Parameters

dwKey

Specifies the AutoDial parameter to retrieve. This parameter can be one of the following values.

Value	Meaning
RASADP_ DisableConnectionQuery	The *lpvValue* parameter returns a **DWORD** value. If this value is zero (the default), AutoDial displays a dialog box to query the user before creating a connection. If this value is 1, and the AutoDial database has the phone book entry to dial, AutoDial creates a connection without displaying the dialog box.
RASADP_ LoginSessionDisable	The *lpvValue* parameter returns a **DWORD** value. If this value is 1, the system disables all AutoDial connections for the current logon session. If this value is zero (the default), AutoDial connections are enabled. The AutoDial system service changes this value to zero when a new user logs on to the workstation.
RASADP_ SavedAddressesLimit	The *lpvValue* parameter returns a **DWORD** value that indicates the maximum number of addresses that AutoDial stores in the registry. AutoDial first stores addresses that it used to create an AutoDial connection; then it stores addresses that it learned after a RAS connection was created. Addresses written using the **RasSetAutodialAddress** function are always saved, and are not included in calculating the limit. The default value is 100.
RASADP_ FailedConnectionTimeout	The *lpvValue* parameter returns a **DWORD** value that indicates a time-out value, in seconds. When an AutoDial connection attempt fails, the AutoDial system service disables subsequent attempts to reach the same address for the time-out period. This prevents AutoDial from displaying multiple connection dialog boxes for the same logical request by an application. The default value is 5.
RASADP_ ConnectionQueryTimeout	The *lpvValue* parameter points to a **DWORD** value that indicates a time-out value, in seconds. Before attempting an AutoDial connection, the system will display a dialog asking the user to confirm that the system should dial. The dialog has a countdown timer that will terminate the dialog with a "Do not dial" selection if the user takes no action. The **DWORD** value pointed to by *lpvValue* specifies the initial time on this countdown timer.

lpvValue
Pointer to a buffer that receives the value for the specified parameter.

lpdwcbValue
Pointer to a **DWORD** value. On input, set this value to indicate the size, in bytes, of the *lpvValue* buffer. On output, this value indicates the actual size of the value written to the buffer.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_INVALID_PARAMETER	The *dwKey* or *lpvValue* parameter is invalid.
ERROR_INVALID_SIZE	The size specified by the *lpdwcbValue* is too small.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasSetAutodialAddress**, **RasSetAutodialParam**

RasGetConnectionStatistics

The **RasGetConnectionStatistics** function retrieves accumulated connection statistics for the specified connection.

```
DWORD RasGetConnectionStatistics(
  HRASCONN hRasConn,        // handle to the connection
  RAS_STATS *lpStatistics   // buffer to receive statistics
);
```

Parameters

hRasConn,
Handle to the connection. Use **RasDial** or **RasEnumConnections** to obtain this handle.

lpStatistics
Pointer to a **RAS_STATS** structure to receive the statistics. Set the **dwSize** member of this structure to **sizeof(RAS_STATS)** before calling **RasGetConnectionStatistics**. This parameter cannot be NULL.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is one of the following error codes.

Value	Meaning
E_INVALID_ARG	At least one of the following is true: the *hRasConn* parameter is zero, the *lpStatistics* parameter is NULL, or the value specified by the **dwSize** member of the **RAS_STATS** structure specifies a version of the structure that is not supported by the operating system in use.
ERROR_NOT_ENOUGH_MEMORY	The function could not allocate sufficient memory to complete the operation.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasClearConnectionStatistics**, **RasDial**, **RasEnumConnections**, **RasGetLinkStatistics**

RasGetConnectStatus

The **RasGetConnectStatus** function retrieves information on the current status of the specified remote access connection. An application can use this call to determine when an asynchronous **RasDial** call is complete.

```
DWORD RasGetConnectStatus(
  HRASCONN hrasconn, // handle to RAS connection of interest
  LPRASCONNSTATUS lprasconnstatus
                    // buffer to receive status data
);
```

Parameters

hrasconn

Specifies the remote access connection for which to retrieve the status. This handle must have been obtained from **RasDial** or **RasEnumConnections**.

lprasconnstatus

Pointer to a **RASCONNSTATUS** structure that the function fills with status information. Before calling the function, an application must set the **dwSize** member of the structure to **sizeof(RASCONNSTATUS)** in order to identify the version of the structure being passed.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is a nonzero error value listed in the RAS header file or either ERROR_BUFFER_TOO_SMALL or ERROR_NOT_ENOUGH_MEMORY.

Remarks

The return value for **RasGetConnectStatus** is not necessarily equal to the value of the **dwError** member of the **RASCONNSTATUS** structure returned by **RasGetConnectStatus**. The return value of **RasGetConnectStatus** indicates errors that occur during the **RasGetConnectStatus** function call, whereas the **dwError** member indicates errors that prevented the connection from being established.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASCONNSTATUS**, **RasDial**, **RasEnumConnections**

RasGetCountryInfo

The **RasGetCountryInfo** function retrieves country-specific dialing information from the Windows Telephony list of countries.

```
DWORD RasGetCountryInfo(
  LPRASCTRYINFO lpRasCtryInfo,
                    // buffer that receives country info
  LPDWORD lpdwSize  // size, in bytes, of the buffer
);
```

For more information about country-specific dialing information and Telephony Application Programming Interface (TAPI) country identifiers, see the *TAPI* portion of the *Platform SDK*.

Parameters

lpRasCtryInfo

Pointer to a **RASCTRYINFO** structure that receives the country-specific dialing information followed by additional bytes for a country description string. Before calling the function, set the **dwSize** member of the structure to `sizeof(RASCTRYINFO)` to identify the version of the structure. You must also set the **dwCountryId** member to the TAPI country identifier of the country for which to get information.

The size of the buffer should be at least 256 bytes.

lpdwSize

Pointer to a variable that contains the size, in bytes, of the buffer pointed to by the *lpRasCtryInfo* parameter. On return, the function sets this variable to the number of bytes required.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_INVALID_USER_BUFFER	The address or buffer specified by *lpRasCtryInfo* is invalid.
ERROR_INVALID_PARAMETER	The **dwCountryId** member of the structure pointed to by *lpRasCtryInfo* was not a valid value.
ERROR_BUFFER_TOO_SMALL	The size of the *lpRasCtryInfo* buffer specified by the *lpdwSize* parameter was not large enough to store the information for the country identified by the **dwCountryId** member. The function returns the required buffer size in the variable pointed to by *lpdwSize*.
ERROR_TAPI_CONFIGURATION	TAPI subsystem information was corrupted.

Remarks

To enumerate information for all countries in the Windows Telephony list, set the **dwCountryId** member of the **RASCTRYINFO** structure to 1 in the initial **RasGetCountryInfo** call. This causes the function to return information for the first country in the list. The value returned in the **dwNextCountryId** member is the country identifier of the next country in the list. Use this value in repeated calls to **RasGetCountryInfo** until **dwNextCountryID** returns zero, indicating the last country in the list.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASCTRYINFO**

RasGetCredentials

The **RasGetCredentials** function retrieves the user credentials associated with a specified RAS phone book entry.

```
DWORD RasGetCredentials(
  LPCTSTR lpszPhonebook,          // pointer to the full
                                  // path andfile name of
                                  // a phone book file
  LPCTSTR lpszEntry,              // pointer to the name
                                  // of a phone book entry
  LPRASCREDENTIALS lpCredentials  // pointer to structure
                                  // that receives
                                  // credentials
);
```

Parameters

lpszPhonebook

Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry
Pointer to a null-terminated string that contains the name of a phone book entry.

lpCredentials
Pointer to a **RASCREDENTIALS** structure that receives the user credentials associated with the specified phone book entry. Before calling **RasGetCredentials**, set the **dwSize** member of the structure to `sizeof(RASCREDENTIALS)`, and set the **dwMask** member to indicate the credential information to retrieve. When the function returns, **dwMask** indicates the members that were successfully retrieved.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_CANNOT_OPEN_PHONEBOOK	The specified phone book cannot be found.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	The specified entry does not exist in the phone book.
ERROR_INVALID_PARAMETER	The *lpCredentials* parameter was NULL.
ERROR_INVALID_SIZE	The **dwSize** member of the **RASCREDENTIALS** structure is an unrecognized value.

Remarks

The **RasGetCredentials** function retrieves the credentials of the last user in order to connect using the specified phone book entry, or the credentials subsequently specified in a call to the **RasSetCredentials** function for the phone book entry.

The **RasGetCredentials** function retrieves the user credentials that are stored securely for the specified phone book entry. This function is the preferred way of securely retrieving the credentials associated with a RAS phone book entry. **RasGetCredentials** supersedes the **RasGetEntryDialParams** function, which may not be supported in future releases of Windows 2000.

Windows 2000 and later versions: RasGetCredentials does not return the actual password. Instead, the **szPassword** member of the **RASCREDENTIALS** structure contains a handle to the saved password. You can substitute this handle for the saved password in subsequent calls to **RasSetCredentials** and **RasDial**. When presented with this handle, RasDial will retrieve and use the saved password The value of this handle may change in future versions of the operating system; do not develop code that depends on the contents or format of this value.

Windows 2000 and later versions: The **dwMask** member of **RASCREDENTIALS** contains the RASCM_Password flag if the system has saved a password for the specified entry. If the system has no password saved for this entry, **dwMask** does not contain RASCM_Password.

The following sample code retrieves the credentials for the phone book entry with the name "mazy":

```
ZeroMemory(&lpCred, sizeof(lpCred));
lpCred.dwSize = sizeof(RASCREDENTIALS);
lpCred.dwMask=RASCM_UserName | RASCM_Password | RASCM_Domain ;
res = RasGetCredentials(NULL, "mazy", &lpCred);
if(res == 0)
    printf("The following credentials were retreived:\n%s\n%s\n%s\n",
      lpCred.szUserName,lpCred.szPassword,lpCred.szDomain);
else
    printf("Error: %u\n",res);
```

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASCREDENTIALS**, **RasGetEntryDialParams**, **RasSetCredentials**

RasGetCustomAuthData

Use the **RasGetCustomAuthData** function to retrieve connection-specific authentication information. This information is not specific to a particular user.

```
DWORD RasGetCustomAuthData (
  LPCWSTR pszPhonebook,        // path to phone book to use
  LPCWSTR pszEntry,            // name of entry in phone book
  BYTE * pbCustomAuthData,     // buffer to receive data
  DWORD * pdwSizeofCustomAuthData    // size of buffer
);
```

Parameters

pszPhonebook

Pointer to a null-terminated string containing the full path of the phone book (PBK) file. If this parameter is NULL, the function will use the system phone book.

pszEntry
Pointer to a null-terminated string containing an existing entry name.

pbCustomAuthData
Pointer to a buffer to receive the authentication data. The caller should allocate the memory for this buffer. If the buffer is not large enough, **RasGetCustomAuthData** will return ERROR_BUFFER_TOO_SMALL, and the *pdwSizeofEapData* parameter will contain the required size.

pdwSizeofCustomAuthData
Pointer to a **DWORD** variable that contains the size of the buffer pointed to by the *pbCustomAuthData* parameter.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is one of the following error codes.

Value	Meaning
E_INVALIDARG	The *pdwSizeofCustomAuthData* parameter is NULL.
ERROR_BUFFER_TOO_SMALL	The buffer pointed to by *pbCustomAuthData* is too small to receive the data. The *pdwSizeofCustomAuthData* contains the required size.
ERROR_CANNOT_OPEN_PHONEBOOK	**RasGetEapUserData** was unable to open the specified phone book file.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	**RasGetEapUserData** was unable to find the specified entry in the phone book.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows 2000.

See Also

RasGetEapUserData, **RasSetCustomAuthData**

RasGetEapUserData

Use the **RasGetEapUserData** function to retrieve user-specific Extensible Authentication Protocol (EAP) information for the specified phone book entry.

```
DWORD RasGetEapUserData (
  HANDLE hToken,              // access token for user
  LPCTSTR pszPhonebook,       // path to phone book to use
  LPCTSTR pszEntry,           // name of entry in phone book
  BYTE *pbEapData,            // retrieved data for the user
  DWORD *pdwSizeofEapData     // size of retrieved data
);
```

Parameters

hToken
Handle to a primary or impersonation access token that represents the user for which to retrieve data. This parameter can be NULL if the function is called from a process already running in the user's context.

pszPhonebook
Pointer to a null-terminated string containing the full path of the phone book (PBK) file. If this parameter is NULL, the function will use the system phone book.

pszEntry
Pointer to a null-terminated string containing an existing entry name.

pbEapData
Pointer to a buffer to receive the retrieved EAP data for the user. The caller should allocate the memory for this buffer. If the buffer is not large enough, **RasGetEapUserData** will return ERROR_BUFFER_TOO_SMALL, and the *pdwSizeofEapData* parameter will contain the required size.

pdwSizeofEapData
Pointer to a **DWORD** variable that contains the size of the buffer pointed to by the *pbEapData* parameter.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is one of the following error codes.

Value	Meaning
E_INVALIDARG	The *pdwSizeofEapData* parameter is NULL.
ERROR_BUFFER_TOO_SMALL	The buffer pointed to by *pbEapData* is too small to receive the data. The *pdwSizeofEapData* contains the required size.

(continued)

(continued)

Value	Meaning
ERROR_CANNOT_OPEN_PHONEBOOK	**RasGetEapUserData** was unable to open the specified phone book file.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	**RasGetEapUserData** was unable to find the specified entry in the phone book.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows 2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasSetEapUserData**, **RASEAPINFO**

RasGetEapUserIdentity

The **RasGetEapUserIdentity** function retrieves identity information for the current user. Use this information to call **RasDial** with a phone book entry that requires Extensible Authentication Protocol (EAP).

```
DWORD RasGetEapUserIdentity(
  LPCSTR pszPhonebook,      // path to phone book to use
  LPCSTR pszEntry,          // name of entry in phone book
  DWORD dwFlags,            // flags that qualify
                            // authentication
  HWND hwnd,                // handle to UI parent
  LPRASEAPUSERIDENTITY * ppRasEapUserIdentity,
                            // identity info
);
```

Parameters

pszPhonebook
Pointer to a null-terminated string containing the full path of the phone book (PBK) file. If this parameter is NULL, the function will use the system phone book.

pszEntry
Pointer to a null-terminated string containing an existing entry name.

dwFlags

Specifies zero or more of the following flags that qualify the authentication process.

Flag	Description
RASEAPF_NonInteractive	Specifies that the authentication protocol should not bring up a graphical user-interface. If this flag is not present, it is okay for the protocol to display a user interface.
RASEAPF_Logon	Specifies that the user data is obtained from Winlogon.
RASEAPF_Preview	Specifies that the user should be prompted for identity information before dialing.

hwnd

Handle to the parent window for the UI dialog. If the *fInvokeUI* parameter is FALSE, then *hwnd* should be NULL.

ppRasEapUserIdentity

Pointer to a pointer that, on successful return, points to a **RASEAPUSERIDENTITY** structure containing EAP user identity information. **RasGetEapUserIdentity** will allocate the memory buffer for the **RASEAPUSERIDENTITY** structure. Free this memory by calling **RasFreeEapUserIdentity**.

Return Values

If the function succeeds, the return value is NO_ERROR.

Otherwise, the function will return one of the following error codes.

Value	Meaning
E_INVALID_ARG	The *pcbEapUserIdentity* parameter is NULL.
ERROR_INTERACTIVE_MODE	The function was called with the RASEAPF_NonInteractive flag. However, the authentication protocol must display a UI in order to obtain the required identity information from the user.
ERROR_INVALID_FUNCTION_FOR_ENTRY	Either the authentication method for this phone book entry is not EAP, or the authentication method is EAP but the protocol uses the standard Windows NT/Windows 2000 credentials dialog to obtain user identity information. In either case, the caller does not need to pass EAP identity information to **RasDial**.
ERROR_RASMAN_CANNOT_INITIALIZE	The Remote Access Service failed to initialize properly.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Remarks

RasGetEapUserIdentiy calls the RAS function **RasGetEapUserData** and the EAP function **RasEapGetIdentity**. **RasEapGetIdentity** is implemented by the authentication protocol.

If the function succeeds, that is the return value is NO_ERROR, the caller should copy the EAP identity information from the structure pointed to by *ppRasEapUserIdentity* to the **RASDIALPARAMS** and **RASDIALEXTENSIONS** structures used in the call to **RasDial**. The following sample code demonstrates how to copy the identity information:

```
lstrcpy(DialParms.szUserName, pRasEapUserIdentity->szUserName);
pDialExts->RasEapInfo.dwSizeofEapInfo = pRasEapUserIdentity->dwSizeofEapInfo;
pDialExts->RasEapInfo.pbEapInfo = pRasEapUserIdentity->pbEapInfo;
```

If the remote access application being developed has a graphical user interface, the caller of **RasGetEapUserIdentity** should not specify the RASEAPF_NonInteractive flag. If the application has a command-line user interface, the caller may want to specify the RASEAPF_NonInteractive flag to prevent the authentication protocol from displaying a graphical user interface.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows 2000.

See Also

RASEAPUSERIDENTITY, **RasDial**, **RasEapGetIdentity**, **RasFreeEapUserIdentity**, **RasGetEapUserData**, **RasSetEapUserData**

RasGetEntryDialParams

The **RasGetEntryDialParams** function retrieves the connection information saved by the last successful call to the **RasDial** or **RasSetEntryDialParams** function for a specified phone book entry.

```
DWORD RasGetEntryDialParams(
  LPCTSTR lpszPhonebook, // pointer to the full path and
                         //  file name of the phone book file
  LPRASDIALPARAMS lprasdialparams,
                        // pointer to a structure that
                        //  receives the connection parameters
```

```
    LPBOOL lpfPassword // indicates whether the user's
                       //  password was retrieved
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

Windows 95: Dial-up networking stores phone book entries in the registry rather than in a phone book file.

lprasdialparams

Pointer to a **RASDIALPARAMS** structure. On input, the **dwSize** member must specify the size of the **RASDIALPARAMS** structure, and the **szEntryName** member must specify a valid phone book entry. On output, the structure receives the connection parameters associated with the specified phone book entry.

Note that the **szPhoneNumber** member of the structure does not receive the phone number associated with the phone book entry. To get the phone number associated with a phone book entry, call the **RasGetEntryProperties** function.

Windows 2000 and later versions: RasGetEntryDialParams does not return the actual password. Instead, the **szPassword** member of the **RASDIALPARAMS** structure contains a handle to the saved password. You can substitute this handle for the saved password in subsequent calls to **RasSetEntryDialParams** and **RasDial**. When presented with this handle, **RasDial** will retrieve and use the saved password. The value of this handle may change in future versions of the operating system; do not develop code that depends on the contents or format of this value.

lpfPassword

Pointer to a flag that indicates whether the function retrieved the password associated with the user name for the phone book entry. The function sets this flag to TRUE if the user's password was returned in the **szPassword** member of the **RASDIALPARAMS** structure pointed to by *lprasdialparams*.

Windows 2000 and later: The *lpfPassword* parameter is TRUE if the system has saved a password for the specified entry. If the system has no password saved for this entry, *lpfPassword* is FALSE.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is one of the following error codes.

Value	Description
ERROR_BUFFER_INVALID	The *lprasdialparams* or *lpfPassword* pointer is invalid, or the *lprasdialparams* buffer is invalid.
ERROR_CANNOT_OPEN_PHONEBOOK	The phone book is corrupted or missing components.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	The phone book entry does not exist.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasDial**, **RASDIALPARAMS**, **RasCreatePhonebookEntry**, **RasEditPhonebookEntry**, **RasSetEntryDialParams**

RasGetEntryProperties

The **RasGetEntryProperties** function retrieves the properties of a phone book entry.

```
DWORD RasGetEntryProperties(
  LPCTSTR lpszPhonebook,     // pointer to full path and
                             //  file name of phone book file
  LPCTSTR lpszEntry,         // pointer to an entry name
  LPRASENTRY lpRasEntry,     // buffer that receives entry
                             // information
  LPDWORD lpdwEntryInfoSize,  // size, in bytes, of the
                              // lpRasEntry buffer
  LPBYTE lpbDeviceInfo,       // buffer that receives
                              // device-specific
                              // configuration information
  LPDWORD lpdwDeviceInfoSize  // size, in bytes, of the
                              // lpbDeviceInfo buffer
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry

Pointer to a null-terminated string containing an existing entry name. If you specify an empty string, "", the function returns default values in the buffers pointed to by the *lpRasEntry* and *lpbDeviceInfo* parameters.

lpRasEntry

Pointer to a **RASENTRY** structure followed by additional bytes for the alternate phone number list, if there is one. The structure receives the connection data associated with the phone book entry specified by the *lpszEntry* parameter. Before calling the function, set the **dwSize** member of the structure to `sizeof(RASENTRY)` to identify the version of the structure. This parameter can be NULL.

lpdwEntryInfoSize

Pointer to a variable that contains the size, in bytes, of the *lpRasEntry* buffer. On return, the function sets this variable to the number of bytes required. This parameter can be NULL if the *lpRasEntry* parameter is NULL.

To determine the required buffer size, call **RasGetEntryProperties** with *lpRasEntry* set to NULL and **lpdwEntryInfoSize* set to zero. The function returns the required buffer size in **lpdwEntryInfoSize*.

lpbDeviceInfo

Pointer to a buffer that receives device-specific configuration information. This is opaque TAPI device configuration information that you should not manipulate directly. This parameter can be NULL. For more information about TAPI device configuration, see the ***lineGetDevConfig*** function in the TAPI Programmer's Reference in the Platform SDK.

Windows NT/2000: This parameter is unused. The calling function should set this parameter to NULL.

lpdwDeviceInfoSize

Pointer to a variable that contains the size, in bytes, of the buffer specified by the *lpbDeviceInfo* parameter. On return, the function sets this variable to the number of bytes required. This parameter can be NULL if the *lpbDeviceInfo* parameter s NULL.

To determine the required buffer size, call **RasGetEntryProperties** with *lpbDeviceInfo* set to NULL and **lpdwDeviceInfoSize* set to zero. The function returns the required buffer size in **lpdwDeviceInfoSize*.

Windows NT/2000: This parameter is unused. The calling function should set this parameter to NULL.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_INVALID_PARAMETER	The function was called with an invalid parameter.
ERROR_INVALID_SIZE	The value of the dwSize member of the *lpRasEntry* is too small.
ERROR_BUFFER_INVALID	The address or buffer specified by *lpRasEntry* is invalid.
ERROR_BUFFER_TOO_SMALL	The buffer size indicated in *lpdwEntryInfoSize* is too small.
ERROR_CANNOT_OPEN_ PHONEBOOK	The phone book is corrupted or is missing components.
ERROR_CANNOT_FIND_ PHONEBOOK_ENTRY	The phone book entry does not exist.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASENTRY**, **RasSetEntryProperties**

RasGetErrorString

The **RasGetErrorString** function obtains an error message string for a specified RAS error value.

```
DWORD RasGetErrorString(
  UINT uErrorValue,         // error to get string for
  LPTSTR lpszErrorString,   // buffer to hold error string
  DWORD cBufSize            // size, in characters, of buffer
);
```

Parameters

uErrorValue

Specifies the error value of interest. These are values returned by one of the RAS functions: those listed in the RAS header file.

lpszErrorString

Pointer to a buffer that the function will write the error string to. This parameter must not be NULL.

cBufSize

Specifies the size, in characters, of the buffer pointed to by *lpszErrorString*.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is a nonzero error value. This value is ERROR_INVALID_PARAMETER or the **GetLastError** value returned from the functions **GlobalAlloc** or **LoadString**. The function does not set a thread's last error information; that is, there is no **GetLastError** information set by the **RasGetErrorString** function.

Remarks

There is no way to determine in advance the exact size in characters of an error message, and thus the size of buffer required. Error messages will generally be 80 characters or fewer in size; a buffer size of 256 characters will always be adequate. A buffer of insufficient size causes the **RasGetErrorString** function to fail, returning ERROR_INSUFFICIENT_BUFFER. Note that buffer sizes are specified in characters, not bytes; thus, the Unicode version of **RasGetErrorString** requires a 512 byte buffer to guarantee that every error message will fit.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **GlobalAlloc**, **LoadString**

RasGetLinkStatistics

The **RasGetLinkStatistics** function retrieves accumulated statistics for the specified link in a RAS multilink connection.

```
DWORD RasGetLinkStatistics (
  HRASCONN hRasConn,       // handle to connection
  DWORD dwSubEntry,        // SubEntry for link
  RAS_STATS *lpStatistics // buffer to receive statistics
);
```

Parameters

hRasConn
Handle to the connection. Use **RasDial** or **RasEnumConnections** to obtain this handle.

dwSubEntry
Specifies the subentry that corresponds to the link for which to retrieve statistics.

lpStatistics
Pointer to a **RAS_STATS** structure to receive the statistics. Set the **dwSize** member of this structure to **sizeof(RAS_STATS)** before calling **RasGetLinkStatistics**. This parameter cannot be NULL.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is one of the following error codes.

Value	Meaning
E_INVALID_ARG	At least one of the following is true: the *hRasConn* parameter is zero, the *dwSubEntry* parameter is zero, the *lpStatistics* parameter is NULL, or the value specified by the **dwSize** member of the **RAS_STATS** structure specifies a version of the structure that is not supported by the operating system in use.
ERROR_NOT_ENOUGH_ MEMORY	The function could not allocate sufficient memory to complete the operation.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasClearLinkStatistics**, **RasDial**, **RasEnumConnections**, **RasGetConnectionStatistics**

RasGetProjectionInfo

The **RasGetProjectionInfo** function obtains information about a remote access projection operation for a specified remote access component protocol.

```
DWORD RasGetProjectionInfo(
  HRASCONN hrasconn,              // handle that specifies
                                  // remote access connection
                                  // of interest
  RASPROJECTION rasprojection,    // specifies type of
                                  // projection information to obtain
  LPVOID lpprojection,            // points to buffer that
                                  // receives projection
                                  // information
  LPDWORD lpcb                    // points to variable that
                                  // specifies buffer size
);
```

Parameters

hrasconn

Handle to the remote access connection of interest. An application obtains a RAS connection handle from the **RasDial** or **RasEnumConnections** function.

rasprojection

Specifies a **RASPROJECTION** enumerated type value that specifies the protocol of interest.

lpprojection

Pointer to a buffer that will receive the information specified by the *rasprojection* parameter. The information will be in a structure appropriate to the *rasprojection* value.

***rasprojection* value**	**Data structure**
RASP_Amb	**RASAMB**
RASP_PppCcp	**RASPPPCCP**
RASP_PppIp	**RASPPPIP**
RASP_PppIpx	**RASPPPIPX**
RASP_PppLcp	**RASPPPLCP**
RASP_PppNbf	**RASPPPNBF**
RASP_Slip	**RASPSLIP**

lpcb

Pointer to a variable that, on entry, specifies the size in bytes of the buffer pointed to by *lpprojection*. On exit, this variable contains the size of the buffer needed to contain the specified projection information.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code. The function may return a nonzero RAS error code, or one of the following error codes.

Value	Meaning
ERROR_BUFFER_TOO_SMALL	The buffer pointed to by *lpprojection* is not large enough to contain the requested information.
ERROR_INVALID_HANDLE	The *hrasconn* parameter is not a valid handle.
ERROR_INVALID_PARAMETER	One of the parameters is invalid.
ERROR_INVALID_SIZE	The **dwSize** member of the structure pointed to by *lpprojection* specifies an invalid size.
ERROR_PROTOCOL_NOT_CONFIGURED	The control protocol for which information was requested neither succeeded nor failed, because the connection's phone book entry did not require that an attempt to negotiate the protocol be made. This is a RAS error code.

Remarks

Remote access projection is the process whereby a remote access server and a remote client negotiate network protocol-specific information. A remote access server uses this network protocol-specific information to represent a remote client on the network.

Windows NT/2000: Remote access projection information is not available until the operating system has executed the **RasDial** RASCS_Projected state on the remote access connection. If **RasGetProjectionInfo** is called prior to the RASCS_Projected state, it returns ERROR_PROJECTION_NOT_COMPLETE.

Windows 95: Windows 95 Dial-Up Networking does not support the RASCS_Projected state. The projection phase may be done during the RASCS_Authenticate state. If the authentication is successful, the connection operation proceeds to the RASCS_Authenticated state, and projection information is available for successfully configured protocols. If **RasGetProjectionInfo** is called prior to the RASCS_Authenticated state, it returns ERROR_PROTOCOL_NOT_CONFIGURED.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASAMB**, **RasDial**, **RasEnumConnections**, **RASPPPNBF**, **RASPPPIPX**, **RASPPPIP**, **RASPROJECTION**

RasGetSubEntryHandle

The **RasGetSubEntryHandle** function retrieves a connection handle for a specified subentry of a multilink connection.

```
DWORD RasGetSubEntryHandle(
  HRASCONN hrasconn,
  DWORD dwSubEntry,
  LPHRASCONN lphrasconn
);
```

Parameters

hrasconn
Specifies an **HRASCONN** connection handle returned by the **RasDial** function for a multilink phone book entry.

dwSubEntry
Specifies a valid subentry index for the phone book entry.

lphrasconn
Pointer to an **HRASCONN** variable that receives a connection handle that represents the subentry connection.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_INVALID_HANDLE	The *hrasconn* connection handle does not represent a connected phone book entry.

(continued)

(continued)

Value	Meaning
ERROR_PORT_NOT_OPEN	The *hrasconn* and *dwSubEntry* parameters are valid, but the specified subentry is not connected.
ERROR_NO_MORE_ITEMS	The value specified by *dwSubEntry* exceeds the maximum number of subentries for the phone book entry.

Remarks

The connection handle specified in the *hrasconn* parameter refers to the entire multilink connection, but the connection handle returned in the **lphrasconn* parameter refers only to the subentry connection. You can use the subentry connection handle in any function that accepts an *hrasconn* parameter, including the **RasHangUp**, **RasGetConnectStatus**, and **RasGetProjectionInfo** functions. The projection information returned by **RasGetProjectionInfo** for a multilink entry is the same for the each of the subentry connection handles as it is for the main connection handle.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasDial**, **RasGetConnectStatus**, **RasGetProjectionInfo**, **RasHangUp**

RasGetSubEntryProperties

The **RasGetSubEntryProperties** function retrieves information about a subentry for a specified phone book entry.

```
DWORD RasGetSubEntryProperties(
  LPCTSTR lpszPhonebook,    // pointer to full path and file
                            // name of phone book file
  LPCTSTR lpszEntry,        // pointer to an entry name
  DWORD dwSubEntry,         // index of the subentry
  LPRASSUBENTRY lpRasSubEntry,
                            // pointer to structure that
                            // receives information about
                            // subentry
```

```
    LPDWORD lpdwcb,           // size, in bytes, of the
                              // structure
    LPBYTE lpbDeviceConfig,   // pointer to TAPI device
                              // configuration
    LPDWORD lpcbDeviceConfig  // pointer to size of device
                              // configuration
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry

Pointer to a null-terminated string containing the name of an existing entry in the phone book.

dwSubEntry

Specifies the one-based index of the subentry.

lpRasSubEntry

Pointer to a **RASSUBENTRY** structure followed by additional bytes for the alternate phone number list, if there is one. The structure receives the information about the specified subentry. Before calling the function, set the **dwSize** member of the structure to `sizeof(RASSUBENTRY)` to identify the version of the structure. This parameter can be NULL.

lpdwcb

Pointer to a variable that contains the size, in bytes, of the *lpRasSubEntry* buffer. On return, the function sets this variable to the number of bytes returned, or the number of bytes required if the buffer is too small. This parameter can be NULL if *lpRasSubEntry* is NULL.

lpbDeviceConfig

Pointer to a TAPI device configuration block. This parameter is currently unused. The caller should pass NULL for this parameter. For more information about TAPI device configuration blocks, see the function **lineGetDevConfig**.

lpcbDeviceConfig

Pointer to a **DWORD** to receive the size of the TAPI device configuration block. This parameter is currently unused. The caller should pass NULL for this parameter.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_INVALID_PARAMETER	The function was called with an invalid parameter.
ERROR_BUFFER_INVALID	The address or buffer specified by *lpRasSubEntry* is invalid.
ERROR_BUFFER_TOO_SMALL	The *lpRasSubEntry* buffer is too small. The *lpdwcb* variable receives the required buffer size.
ERROR_CANNOT_OPEN_PHONEBOOK	The phone book is corrupted or is missing components.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	The phone book entry does not exist.

Remarks

A RAS phone book entry can have zero or more subentries, each minimally consisting of a device and a phone number. A phone book entry with multiple subentries can be configured to dial the first available or all subentries when the entry is dialed.

Use the **RasGetEntryProperties** function to retrieve a **RASENTRY** structure containing information about the subentries of a phone book entry. The **dwSubEntries** member indicates the number of subentries and the **dwDialMode** member indicates the dialing configuration.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasGetEntryProperties**, **RASENTRY**, **RasSetSubEntryProperties**, **RASSUBENTRY**

RasHangUp

The **RasHangUp** function terminates a remote access connection. The connection is specified with a RAS connection handle. The function releases all RASAPI32.DLL resources associated with the handle.

```
DWORD RasHangUp(
  HRASCONN hrasconn   // handle to the RAS connection to hang up
);
```

Parameters

hrasconn

Specifies the remote access connection to terminate. This is a handle returned from a previous call to **RasDial** or **RasEnumConnections**.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is a nonzero error value listed in the RAS header file, or ERROR_INVALID_HANDLE.

Remarks

The connection is terminated even if the **RasDial** call has not yet been completed.

After this call, the *hrasconn* handle can no longer be used.

An application should not call **RasHangUp** and then immediately exit. The connection state machine needs time to properly terminate. If the system prematurely terminates the state machine, the state machine may fail to properly close a port, leaving the port in an inconsistent state. A simple way to avoid this problem is to call **Sleep**(3000) after returning from **RasHangUp**; after that pause, the application can exit. A more responsive way to avoid the problem is, after returning from **RasHangUp**, to call **RasGetConnectStatus**(*hrasconn*) and **Sleep**(0) in a loop until **RasGetConnectStatus** returns ERROR_INVALID_HANDLE.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASCONN**, **RasCustomHangUp**, **RasDial**, **RasEnumConnections**, **RasGetConnectStatus**, **Sleep**

RasInvokeEapUI

The **RasInvokeEapUI** function displays a custom user interface to obtain Extensible Authentication Protocol (EAP) information from the user.

```
DWORD RasInvokeEapUI(
  HRASCONN hRasConn,   // handle to connection
  DWORD dwSubEntry,    // subentry from callback
  LPRASDIALEXTENSIONS lpExtensions,
  HWND hwnd            // handle to parent window for
                       // user interface
);
```

Parameters

hRasConn
Handle to the connection returned by **RasDial**.

dwSubEntry
Specifies the subentry returned in the callback.

lpExtensions
Pointer to a **RASDIALEXTENSIONS** structure. This structure should be the same as that passed to **RasDial** when restarting from a paused state. The **dwSize** member of the **RASDIALEXTENSIONS** structure must be set to **sizeof(RASDIALEXTENSIONS)**. This parameter cannot be NULL.

hwnd
Handle to the parent window to use when displaying the EAP user interface.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_INVALID_ HANDLE	The *hRassConn* parameter is zero, or the *lpExtensions* parameter is NULL.
ERROR_INVALID_SIZE	The value of the **dwSize** member of the **RASDIALEXTENSIONS** structure specifies a version of the structure that isn't supported by the operating system in use.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasDial**, **RASDIALEXTENSIONS**, **RASEAPINFO**

RasMonitorDlg

The **RasMonitorDlg** function displays the **Dial-Up Networking Monitor** property sheet that describes the status of RAS connections.

```
BOOL RasMonitorDlg(
  LPTSTR lpszDeviceName,  // pointer to the name of the
                          // device to display initially
  LPRASMONITORDLG lpInfo  // pointer to structure that
                          // contains input and output
                          // parameters
);
```

Parameters

lpszDeviceName
Pointer to a null-terminated string that specifies the name of the device to display initially. If this parameter is NULL, or if the specified device does not exist, the property sheet displays the first device.

lpInfo
Pointer to a **RASMONITORDLG** structure that contains additional input and output parameters. On input, the **dwSize** member of this structure must specify **sizeof**(**RASMONITORDLG**). If an error occurs, the **dwError** member returns an error code; otherwise, it returns zero.

Return Values

If the user hangs up a connection, the return value is a nonzero value.

If an error occurs, or if the user closes the dialog box without hanging up a connection, the return value is zero. If an error occurs, the **dwError** member of the **RASMONITORDLG** structure returns a nonzero system error code or RAS error code.

Remarks

The following sample code invokes the RAS monitor dialog:

```
lpInfo = (LPRASMONITORDLG)GlobalAlloc(GPTR, sizeof(RASMONITORDLG));

ZeroMemory(lpInfo, sizeof(RASMONITORDLG));
// Essential, since garbage values cause the API to fail
lpInfo->dwSize=sizeof(RASMONITORDLG);
```

(continued)

(continued)

```
nRet = RasMonitorDlg(NULL,lpInfo);

if (nRet)
    printf("User hung up the connection\n");
else
{
    if (lpInfo->dwError != 0)
    {
        printf("RasMonitorDlg failed: Error = %d\n", lpInfo->dwError);
        return -1;
    }
    else
        printf("User pressed Close\n");
}
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Library: Use Rasdlg.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASMONITORDLG**

RasPBDlgFunc

The **RasPBDlgFunc** function is an application-defined callback function that receives notifications of user activity while the **RasPhonebookDlg** dialog box is open.

```
VOID WINAPI RasPBDlgFunc(
  DWORD dwCallbackId,  // an application-defined value
  DWORD dwEvent,       // indicates the event that occurred
  LPTSTR pszText,      // pointer to an additional
                       // string argument
  LPVOID pData         // pointer to an additional
                       // buffer argument
);
```

Parameters

dwCallbackId

Specifies the application-defined value that was specified in the **dwCallback** member of the **RASPBDLG** structure passed to the **RasPhonebookDlg** function.

dwEvent

A set of bit flags that indicates the event that occurred. This parameter is one of the following values.

Value	Meaning
RASPBDEVENT_AddEntry	Received when the user creates a new phone book entry or copies an existing phone book entry. The *pszText* parameter is the name of the new or copied entry. The *pData* parameter is undefined.
RASPBDEVENT_EditEntry	Received when the user changes an existing phone book entry. The *pszText* parameter is the name of the modified entry. The *pData* parameter is undefined.
RASPBDEVENT_RemoveEntry	Received when the user deletes a phone book entry. The *pszText* parameter is the name of the deleted entry. The *pData* parameter is undefined.
RASPBDEVENT_DialEntry	Received when the user successfully dials an entry. The *pszText* parameter is the name of the newly connected entry. The *pData* parameter is undefined.
RASPBDEVENT_EditGlobals	Received when the user makes changes in the **User Preferences** property sheet. The *pszText* parameter is the full path of the default phone book file selected by the user. The *pData* parameter is undefined. This event is also received during dialog startup if the *lpszPhonebook* parameter of the **RasPhonebookDlg** call is NULL. In this case, the event informs the caller of the path of the default phone book.
RASPBDEVENT_NoUser	Received during dialog box initialization when the RASPBDFLAG_NoUser flag is set. The *pData* parameter is a pointer to a **RASNOUSER** structure. The callback function should fill the structure with the user's logon credentials and dialog time out. The **RasPhonebookDlg** function then uses the supplied credentials for authentication by the remote server. The *pszText* parameter is undefined.
RASPBDEVENT_NoUserEdit	Received if the RASPBDFLAG_NoUser flag is set and the user changes the credentials that you supplied during the RASPBDEVENT_NoUser event. The *pData* parameter is a pointer to a **RASNOUSER** structure containing the updated credentials. This occurs during a dialing operation if the user changes his or her password, or if the authentication fails and the user retries authentication with different credentials. The *pszText* parameter is undefined.

pszText

Pointer to an additional string argument whose meaning depends on the event indicated in the *dwEvent* parameter.

pData

Pointer to an additional buffer argument whose meaning depends on the event indicated in the *dwEvent* parameter.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Unicode: Declared as Unicode and ANSI prototypes.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASNOUSER**, **RasPhonebookDlg**

RasPhonebookDlg

The **RasPhonebookDlg** function displays the main **Dial-Up Networking** dialog box. From this modal dialog box, the user can dial, edit, or delete a selected phone book entry, create a new phone book entry, or specify user preferences. The **RasPhonebookDlg** function returns when the dialog box closes.

```
BOOL RasPhonebookDlg(
  LPTSTR lpszPhonebook,  // pointer to the full path and
                         // file name of the phone book file
  LPTSTR lpszEntry,      // pointer to the name of the
                         // phone book entry to highlight
  LPRASPBDLG lpInfo      // pointer to a structure that
                         // contains additional parameters
);
```

Parameters

lpszPhonebook

Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry

Pointer to a null-terminated string that contains the name of the phone book entry to highlight initially. If this parameter is NULL, or if the specified entry does not exist, the dialog box highlights the first entry in the alphabetic list.

lpInfo

Pointer to a **RASPBDLG** structure that contains additional input and output parameters. On input, the **dwSize** member of this structure must specify **sizeof(RASPBDLG)**. If an error occurs, the **dwError** member returns an error code; otherwise, it returns zero.

Return Values

If the user selects the **Dial** button and the function establishes a connection, the return value is a nonzero value.

If an error occurs, or if the user selects the **Close** button to close the dialog box, the return value is zero. If an error occurs, the **dwError** member of the **RASPBDLG** structure returns a nonzero system error code or RAS error code.

The following sample code brings up the **Dial-Up Networking** dialog. The dialog will display dialing information for the first entry from the default phone book file.

```
lpInfo = (LPRASPBDLG)GlobalAlloc(GPTR, sizeof(RASPBDLG));
// Essential, since garbage values cause the API to fail
ZeroMemory(lpInfo, sizeof(RASPBDLG));
lpInfo->dwSize=sizeof(RASPBDLG);

nRet = RasPhonebookDlg(NULL,NULL,lpInfo);

if (nRet)
    printf("User pressed Dial\n");
else
{
    if (lpInfo->dwError != 0)
    {
        printf("RasPhonebookDlg failed: Error = %d\n", lpInfo->dwError);
    }
    else
        printf("User pressed Close\n");
}
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Library: Use Rasdlg.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASPBDLG**

RasRenameEntry

The **RasRenameEntry** function changes the name of an entry in a phone book.

```
DWORD RasRenameEntry(
  LPCTSTR lpszPhonebook, // pointer to full path and file
                         // name of phone book file
  LPCTSTR lpszOldEntry,  // pointer to the old entry name
  LPCTSTR lpszNewEntry   // pointer to the new entry name
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

Windows 95: This parameter should always be NULL. Dial-up networking stores phone book entries in the registry rather than in a phone book file.

lpszOldEntry

Pointer to a null-terminated string containing an existing entry name.

lpszNewEntry

Pointer to a null-terminated string containing the new entry name. Before calling **RasRenameEntry**, call the **RasValidateEntryName** function to validate the new entry name.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_INVALID_NAME	The *lpszNewEntry* name is invalid.
ERROR_ALREADY_EXISTS	An entry with the *lpszNewEntry* name already exists.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	The phone book entry does not exist.

Remarks

The **RasRenameEntry** function allows entry names that would not be accepted by the dial-up networking user interface. The entry names specified in **RasRenameEntry** can consist of any string that adheres to the following conditions.

1. The string cannot have a length greater than RAS_MaxEntryName (as defined in Ras.h).
2. The string cannot consist entirely of space or tab characters.
3. The first character in the string cannot be a period character (".").

The following code sample renames the phone book entry with the name specified by *lpszOldEntry* to the new name specified by *lpszNewEntry*:

```
nRet = RasRenameEntry(NULL, lpszOldEntry, lpszNewEntry);
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasValidateEntryName**

RasSetAutodialAddress

The **RasSetAutodialAddress** function can add an address to the AutoDial mapping database. Alternatively, the function can delete or modify the data associated with an existing address in the database.

```
DWORD RasSetAutodialAddress(
  LPCTSTR lpszAddress,       // pointer to a network
                             // address string
  DWORD dwReserved,          // reserved; must be zero
  LPRASAUTODIALENTRY lpAutoDialEntries,
                             // pointer to buffer containing
                              // AutoDial entry data
  DWORD dwcbAutoDialEntries, // size,in bytes, of the buffer
  DWORD dwcAutoDialEntries    // number of entries in buffer
);
```

Parameters

lpszAddress

Pointer to a null-terminated string that specifies the address to add, delete, or modify. This can be an IP address, Internet host name ("www.microsoft.com"), or NetBIOS name ("products1").

dwReserved
Reserved; must be zero.

lpAutoDialEntries
Pointer to an array of one or more **RASAUTODIALENTRY** structures to be associated with the *lpszAddress* address. If *lpAutoDialEntries* is NULL and *dwcbAutodialEntries* is zero, **RasSetAutodialAddress** deletes all structures associated with *lpszAddress* from the mapping database.

dwcbAutoDialEntries
Specifies the size, in bytes, of the *lpAutodialEntries* buffer.

dwcAutoDialEntries
Specifies the number of **RASAUTODIALENTRY** structures in the *lpAutoDialEntries* buffer.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_INVALID_SIZE	The **dwSize** member of the **RASAUTODIALENTRY** structure is an invalid value.
ERROR_INVALID_PARAMETER	The *lpszAddress* parameter was NULL.

Remarks

An address in the AutoDial mapping database can have any number of associated **RASAUTODIALENTRY** entries. Each entry specifies AutoDial information for a particular TAPI dialing location.

If the address specified by the *lpszAddress* parameter is an existing address in the database and the *lpAutoDialEntries* parameter is not NULL, the **RasSetAutodialAddress** function modifies the set of AutoDial entries associated with the address. If an entry in the *lpAutoDialEntries* array specifies a dialing location for which the address already has an entry, the function replaces the existing entry with the new entry. Otherwise, the function simply adds the *lpAutoDialEntries* entries to the set of entries for the address.

If the *lpszAddress* address exists in the database and *lpAutoDialEntries* is NULL and *dwcbAutodialEntries* is zero, **RasSetAutodialAddress** deletes the address from the database.

If the *lpszAddress* address does not exist in the database, **RasSetAutodialAddress** adds the address to the database. The *lpAutoDialEntries* parameter specifies the AutoDial entries to associate with the new address.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASAUTODIALENTRY**, **RasEnumAutodialAddresses**, **RasGetAutodialAddress**

RasSetAutodialEnable

The **RasSetAutodialEnable** function enables or disables the AutoDial feature for a specified TAPI dialing location. For more information about TAPI dialing locations, see the *(TAPI) Programmer's Reference* in the *Platform SDK documentation*.

```
DWORD RasSetAutodialEnable(
  DWORD dwDialingLocation,
               // identifier of the TAPI dialing location
  BOOL fEnabled  // AutoDial state for this location
);
```

Parameters

dwDialingLocation
Specifies the identifier of a TAPI dialing location.

fEnabled
Specify TRUE to enable AutoDial for the specified dialing location, or FALSE to disable it.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is a nonzero error code.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasGetAutodialEnable**

RasSetAutodialParam

The **RasSetAutodialParam** function sets the value of an AutoDial parameter:

```
DWORD RasSetAutodialParam(
  DWORD dwKey,       // indicates the parameter to set
  LPVOID lpvValue,  // pointer to a buffer that specifies
                     // the value
  DWORD dwcbValue   // size, in bytes, of the buffer
);
```

Parameters

dwKey

Specifies the AutoDial parameter to set. This parameter can be one of the following values.

Value	Meaning
RASADP_ DisableConnectionQuery	The *lpvValue* parameter points to a **DWORD** value. If this value is zero (the default), AutoDial displays a dialog box to query the user before creating a connection. If this value is 1, and the AutoDial database has the phone book entry to dial, AutoDial creates a connection without displaying the dialog box.
RASADP_ LoginSessionDisable	The *lpvValue* parameter points to a **DWORD** value. If this value is 1, the system disables all AutoDial connections for the current logon session. If this value is zero (the default), AutoDial connections are enabled. The AutoDial system service changes this value to zero when a new user logs on to the workstation.

Value	Meaning
RASADP_ SavedAddressesLimit	The *lpvValue* parameter points to a **DWORD** value that indicates the maximum number of addresses that AutoDial stores in the registry. AutoDial first stores addresses that it used to create an AutoDial connection; then it stores addresses that it learned after a RAS connection was created. Addresses written using the **RasSetAutodialAddress** function are always saved, and are not included in calculating the limit. The default value is 100.
RASADP_ FailedConnectionTimeout	The *lpvValue* parameter points to a **DWORD** value that indicates a time-out value, in seconds. When an AutoDial connection attempt fails, the AutoDial system service disables subsequent attempts to reach the same address for the time-out period. This prevents AutoDial from displaying multiple connection dialog boxes for the same logical request by an application. The default value is 5.
RASADP_ ConnectionQueryTimeout	The *lpvValue* parameter points to a **DWORD** value that indicates a time-out value, in seconds. Before attempting an AutoDial connection, the system will display a dialog asking the user to confirm that the system should dial. The dialog has a countdown timer that will terminate the dialog with a "Do not dial" selection if the user takes no action. The **DWORD** value pointed to by lpvValue specifies the initial time on this countdown timer.

lpvValue
Pointer to a buffer that contains the new value for the specified parameter.

dwcbValue
Specifies the size, in bytes, of the value in the *lpvValue* buffer.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_INVALID_PARAMETER	The *dwKey* or *lpvValue* parameter is invalid.
ERROR_INVALID_SIZE	The size specified by the *dwcbValue* is invalid.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasGetAutodialParam**, **RasSetAutodialAddress**

RasSetCredentials

The **RasSetCredentials** function sets the user credentials associated with a specified RAS phone book entry.

```
DWORD RasSetCredentials(
  LPCTSTR lpszPhonebook,    // pointer to the full path and
                            // file name of a phone book file
  LPCTSTR lpszEntry,        // pointer to the name of a
                            // phone book entry
  LPRASCREDENTIALS lpCredentials,
                            // pointer to structure that
                            // specifies the credentials
  BOOL fClearCredentials    // if true, credentials are
                            // cleared if false, credentials
                            // are set
);
```

Parameters

lpszPhonebook
Pointer to a null-terminated string that specifies the full path and file name of a phone book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry
Pointer to a null-terminated string that contains the name of a phone book entry.

lpCredentials
Pointer to a **RASCREDENTIALS** structure that specifies the user credentials to set for the specified phone book entry. Before calling **RasSetCredentials**, set the **dwSize** member of the structure to `sizeof(RASCREDENTIALS)`. Set the **dwMask** member to indicate the credential information to be set.

fClearCredentials

Specifies a flag that indicates whether **RasSetCredentials** clears existing credentials by setting them to the empty string, "". If this flag is TRUE, the **dwMask** member of the **RASCREDENTIALS** structure indicates the credentials that the function sets to the empty string. If this flag is FALSE, the function sets the indicated credentials according to the contents of their corresponding **RASCREDENTIALS** members.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_CANNOT_OPEN_ PHONEBOOK	The specified phone book cannot be found.
ERROR_CANNOT_FIND_ PHONEBOOK_ENTRY	The specified entry does not exist in the phone book.
ERROR_INVALID_PARAMETER	The *lpCredentials* parameter was NULL.
ERROR_INVALID_SIZE	The **dwSize** member of the **RASCREDENTIALS** structure is an unrecognized value.

Remarks

The **RasSetCredentials** function sets the user credentials associated with a specified RAS phone book entry. The credentials stored with a phone book entry are the credentials of the last user to successfully connect using the specified phone book entry, or the credentials subsequently specified in a call to the **RasSetCredentials** or **RasSetEntryDialParams** function for the phone book entry.

The **RasSetCredentials** function is the preferred way of securely storing credentials with a phone book entry. **RasSetCredentials** supersedes the **RasSetEntryDialParams** function, which may not be supported in future releases of Windows 2000.

Windows 2000 and later versions: If the **szPassword** member of the **RASCREDENTIALS** structure contains the password handle returned by **RasGetCredentials** or **RasGetEntryDialParams**, **RasSetCredentials** returns successfully without changing any currently saved password.

The following code sample sets the credentials for the phone book entry with the name "mazy".

```
ZeroMemory(&lpCred, sizeof(lpCred));

lpCred.dwSize = sizeof(lpCred);
lstrcpy(lpCred.szUserName, "test");
lstrcpy(lpCred.szPassword, "");
```

(continued)

(continued)

```
lstrcpy(lpCred.szDomain, "BANANA40");
lpCred.dwMask=RASCM_UserName | RASCM_Password | RASCM_Domain ;

res=RasSetCredentials(NULL, "mazy", &lpCred, 0);
if(res == 0)
    printf("Set Credentials to:\n%s\n%s\n%s\n\n",
      lpCred.szUserName,lpCred.szPassword,lpCred.szDomain);
else
    printf("Error: %u\n\n,",res);
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASCREDENTIALS**, **RasGetCredentials**, **RasSetEntryDialParams**

RasSetCustomAuthData

Use the **RasSetCustomAuthData** function to set connection-specific authentication information. This information should not be specific to a particular user.

```
DWORD RasSetCustomAuthData (
  LPCWSTR pszPhonebook,         // path to phone book to use
  LPCWSTR pszEntry,             // name of entry in phone book
  BYTE * pbCustomAuthData,      // pointer to data
  DWORD * dwSizeofCustomAuthData // size of data
);
```

Parameters

pszPhonebook
: Pointer to a null-terminated string containing the full path of the phone book (PBK) file. If this parameter is NULL, the function will use the system phone book.

pszEntry
: Pointer to a null-terminated string containing an existing entry name.

pbCustomAuthData
: Pointer to a buffer containing the new authentication data.

dwSizeofCustomAuthData
: Size of the data pointed to by the *pbCustomAuthData* parameter.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is one of the following error codes.

Value	Meaning
E_INVALIDARG	The *dwSizeofCustomAuthData* parameter is zero, or the *pbCustomAuthData* parameter is NULL.
ERROR_CANNOT_OPEN_PHONEBOOK	**RasSetEapUserData** was unable to open the specified phone book file.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	**RasSetEapUserData** was unable to find the specified entry in the phone book.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows 2000.

See Also

RasGetCustomAuthData, RasSetEapUserData

RasSetEapUserData

Use the **RasSetEapUserData** function to store user-specific Extensible Authentication Protocol (EAP) information for the specified phone book entry in the registry.

```
DWORD RasSetEapUserData(
  HANDLE hToken,           // access token for user
  LPCTSTR pszPhonebook,    // path to phone book to use
  LPCTSTR pszEntry,        // name of entry in phone book
  BYTE *pbEapData,         // data to store for the user
  DWORD dwSizeofEapData,   // size of data
);
```

Parameters

hToken
Handle to a primary or impersonation access token that represents the user for which to store data. This parameter can be NULL if the function is called from a process already running in the user's context.

pszPhonebook
Pointer to a null-terminated string containing the full path of the phone book (PBK) file. If this parameter is NULL, the function will use the system phone book.

pszEntry
Pointer to a null-terminated string containing an existing entry name.

pbEapData
Pointer to the data to store for the user.

dwSizeofEapData
Specifies the size of the data pointed to by the *pbEapData* parameter.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is one of the following error codes.

Value	Meaning
E_INVALIDARG	The *dwSizeofEapData* parameter is zero, or the *pbEapData* parameter is NULL.
ERROR_CANNOT_OPEN_PHONEBOOK	**RasSetEapUserData** was unable to open the specified phone book file.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	**RasSetEapUserData** was unable to find the specified entry in the phone book.
Other	Use **FormatMessage** to retrieve the system error message that corresponds to the error code returned.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows 2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasGetEapUserData**, **RasInvokeEapUI**

RasSetEntryDialParams

The **RasSetEntryDialParams** function changes the connection information saved by the last successful call to the **RasDial** or **RasSetEntryDialParams** function for a specified phone book entry.

```
DWORD RasSetEntryDialParams(
  LPCTSTR lpszPhonebook,  // pointer to the full path and
                          // file name of the phone book file
  LPRASDIALPARAMS lprasdialparams,
                         // pointer to a structure with the
                          // new connection parameters
  BOOL fRemovePassword    // indicates whether to remove
                          // password from entry's parameters
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

Windows 95: Dial-up networking stores phone book entries in the registry rather than in a phone book file.

lprasdialparams

Pointer to a **RASDIALPARAMS** structure containing the connection parameters to be associated with the phone book entry. **RasSetEntryDialParams** uses the structure's members as follows.

Member	Description
dwSize	Must specify the **sizeof(RASDIALPARAMS)** to identify the version of the structure.
szEntryName	A null-terminated string that identifies the phone book entry to set parameters for.
szPhoneNumber	Not used. Set to NULL.
szCallbackNumber	A null-terminated string containing the callback phone number. If **szCallbackNumber** is an empty string (""), the callback number is not changed.
szUserName	A null-terminated string containing the logon name of the user associated with this entry. If **szUserName** is an empty string, the user name is not changed.

(continued)

(continued)

Member	Description
szPassword	A null-terminated string containing the password for the user specified by **szUserName**. If **szUserName** is an empty string, the password is not changed. If **szPassword** is an empty string and *fRemovePassword* is FALSE, the password is set to the empty string. If *fRemovePassword* is TRUE, the password stored in this phone book entry for the user specified by **szUserName** is removed regardless of the contents of the **szPassword** string. **Windows NT 4.0 and later:** The password is changed to the string specified by **szPassword** regardless of whether **szUserName** is an empty string. **Windows 2000 and later:** If **szPassword** contains the password handle returned by **RasGetCredentials** or **RasGetEntryDialParams**, **RasSetEntryDialParams** returns successfully without changing any currently saved password.
szDomain	A null-terminated string containing the name of the domain to log on to. If **szDomain** is an empty string, the domain name is not changed.
dwSubEntry	Specifies the (one-based) index of the initial subentry to dial when establishing the connection.
dwCallbackId	Not used; should be zero.

fRemovePassword

Specifies whether to remove the phone book entry's stored password for the user specified by *lprasdialparams*->**szUserName**. If *fRemovePassword* is TRUE, the password is removed. Setting fRemovePassword to TRUE is equivalent to checking the "Unsave Password" checkbox in Dial-Up Networking. When setting the password or other properties of a phone book entry, set fRemovePassword to FALSE.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Description
ERROR_BUFFER_INVALID	The address or buffer specified by *lprasdialparams* is invalid.
ERROR_CANNOT_OPEN_ PHONEBOOK	The phone book is corrupted or missing components.
ERROR_CANNOT_FIND_ PHONEBOOK_ENTRY	The phone book entry does not exist.

Remarks

To create a new phone book entry, use the **RasSetEntryProperties** function.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASDIALPARAMS**, **RasCreatePhonebookEntry**, **RasEditPhonebookEntry**, **RasGetEntryDialParams**, **RasSetEntryProperties**

RasSetEntryProperties

The **RasSetEntryProperties** function changes the connection information for an entry in the phone book or creates a new phone book entry.

```
DWORD RasSetEntryProperties(
  LPCTSTR lpszPhonebook,  // pointer to full path and file
                          // name of phone book file
  LPCTSTR lpszEntry,      // pointer to an entry name
  LPRASENTRY lpRasEntry,  // buffer that contains entry
                          // information
  DWORD dwEntryInfoSize,  // size, in bytes, of the
                          // lpRasEntry buffer
  LPBYTE lpbDeviceInfo,   // buffer that contains device-
                          // specific configuration
                          // information
  DWORD dwDeviceInfoSize  // size, in bytes, of the
                          // lpbDeviceInfo buffer
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry

Pointer to a null-terminated string containing an entry name.

If the entry name matches an existing entry, **RasSetEntryProperties** modifies the properties of that entry.

If the entry name does not match an existing entry, **RasSetEntryProperties** creates a new phone book entry. For new entries, call the **RasValidateEntryName** function to validate the entry name before calling **RasSetEntryProperties**.

lpRasEntry

Pointer to a **RASENTRY** structure that contains the new connection data to be associated with the phone book entry specified by the *lpszEntry* parameter.

The structure might be followed by an array of null-terminated alternate phone number strings. The last string is terminated by two consecutive null characters. The **dwAlternateOffset** member of the **RASENTRY** structure contains the offset to the first string.

dwEntryInfoSize

Specifies the size, in bytes, of the buffer specified by the *lpRasEntry* parameter.

lpbDeviceInfo

Pointer to a buffer containing device-specific configuration information. This is opaque TAPI device configuration information. For more information about TAPI device configuration, see the ***lineGetDevConfig*** function in the TAPI Programmer's Reference in the Platform SDK.

Windows NT/2000: This parameter is unused. The calling function should set this parameter to NULL.

dwDeviceInfoSize

Specifies the size, in bytes, of the *lpbDeviceInfo* buffer.

Windows NT/2000: This parameter is unused. The calling function should set this parameter to zero.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_BUFFER_INVALID	The address or buffer specified by *lpRasEntry* is invalid.
ERROR_CANNOT_OPEN_PHONEBOOK	The phone book is corrupted or missing components.

Remarks

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RASENTRY**, **RasCreatePhonebookEntry**, **RasGetEntryProperties**, **RasValidateEntryName**

RasSetSubEntryProperties

The **RasSetSubEntryProperties** function creates a new subentry or modifies an existing subentry of a specified phone book entry.

```
DWORD RasSetSubEntryProperties(
  LPCTSTR lpszPhonebook,    // pointer to full path and file
                            // name of the phone book file
  LPCTSTR lpszEntry,        // pointer to an entry name
  DWORD dwSubEntry,         // index of the subentry
  LPRASSUBENTRY lpRasSubEntry,
                            // pointer to structure
                            // containing information about
  DWORD dwcbRasSubEntry     // subentry size, in bytes,
                            // of the structure
  LPBYTE lpbDeviceConfig,   // pointer to TAPI device
                            // configuration
  DWORD dwcbDeviceConfig    // size of TAPI device
                            // configuration
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry
Pointer to a null-terminated string containing the name of an existing entry in the phone book.

dwSubEntry
Specifies the one-based index of the subentry. If the index matches an existing subentry index, the function changes the properties of that subentry. If the index does not match an existing index, the function creates a new subentry.

lpRasSubEntry
Pointer to a **RASSUBENTRY** structure that contains the data for the subentry.

The structure might be followed by an array of null-terminated alternate phone number strings. The last string is terminated by two consecutive null characters. The **dwAlternateOffset** member of the **RASSUBENTRY** structure contains the offset to the first string.

dwcbRasSubEntry
Specifies the size, in bytes, of the *lpRasSubEntry* buffer.

lpbDeviceConfig
Pointer to a TAPI device configuration block. This parameter is currently unused. The caller should pass NULL for this parameter. For more information about TAPI device configuration blocks, see the function ***lineGetDevConfig***.

dwcbDeviceConfig
Specifies the size of the TAPI device configuration block. This parameter is currently unused. The caller should pass zero for this parameter.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_BUFFER_INVALID	The address or buffer specified by *lpRasEntry* is invalid.
ERROR_CANNOT_FIND_PHONEBOOK_ENTRY	The phone book entry does not exist.
ERROR_CANNOT_OPEN_PHONEBOOK	The phone book is corrupted or missing components.
ERROR_INVALID_PARAMETER	The function was called with an invalid parameter.

Remarks

A RAS phone book entry can have zero or more subentries, each minimally consisting of a device and a phone number. A phone book entry with multiple subentries can be configured to dial either the first available subentry or all subentries when the entry is dialed.

Use the **RasGetEntryProperties** function to retrieve a **RASENTRY** structure containing information about the subentries of a phone book entry. The **dwSubEntries** member indicates the number of subentries and the **dwDialMode** member indicates the dialing configuration.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasGetEntryProperties**, **RASENTRY**, **RASSUBENTRY**

RasValidateEntryName

The **RasValidateEntryName** function validates the format of a connection entry name. The name must contain at least one non-white-space alphanumeric character.

```
DWORD RasValidateEntryName(
  LPCTSTR lpszPhonebook,  // pointer to full path and file
                          //  name of phone book file
  LPCTSTR lpszEntry       // pointer to the entry name to
                          // validate
);
```

Parameters

lpszPhonebook

Windows NT/2000: Pointer to a null-terminated string that specifies the full path and file name of a Phone Book (PBK) file. If this parameter is NULL, the function uses the current default phone book file. The default phone book file is the one selected by the user in the **User Preferences** property sheet of the **Dial-Up Networking** dialog box.

lpszEntry

Pointer to a null-terminated string containing an entry name.

Windows NT/2000: The entry name cannot begin with a period (".").

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is ERROR_INVALID_NAME or ERROR_ALREADY_EXISTS.

Remarks

The following sample code validates the phone book entry specified by the variable *lpszEntry*.

```
nRet = RasValidateEntryName(NULL, lpszEntry);
switch (nRet)
{
    case ERROR_SUCCESS:
        printf("Entry name: %s is valid but doesn't exist in the default phone
book\n", lpszEntry);
        break;
    case ERROR_INVALID_NAME:
        printf("Entry name: %s is invalid\n", lpszEntry);
        break;
    case ERROR_ALREADY_EXISTS:
        printf("Entry name: %s already exists in the default phone book\n",
lpszEntry);
        break;
    default:
        printf("RasValidateEntryName failed: Error = %d\n", nRet);
        break;
}
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Library: Use Rasapi32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Functions, **RasCreatePhonebookEntry**, **RasGetEntryProperties**

RAS Custom Scripting DLL Functions

Implement the following functions when developing a RAS custom-scripting DLL:

- **RasCustomScriptExecute**
- **RasGetBuffer**
- **RasFreeBuffer**
- **RasSendBuffer**
- **RasReceiveBuffer**
- **RasRetrieveBuffer**

RasCustomScriptExecute

RAS calls the **RasCustomScriptExecute** function when establishing a connection for a phone book entry that has the RASEO_CustomScript option set.

```
DWORD RasCustomScriptExecute(
  HANDLE                hPort,
  LPCWSTR               lpszPhonebook,
  LPCWSTR               lpszEntryName,
  PFNRASGETBUFFER       pfnRasGetBuffer,
  PFNRASFREEBUFFER      pfnRasFreeBuffer,
  PFNRASSENDBUFFER      pfnRasSendBuffer,
  PFNRASRECEIVEBUFFER   pfnRasReceiveBuffer,
  PFNRASRETRIEVEBUFFER  pfnRasRetrieveBuffer,
  HWND                  hWnd
  RASDIALPARAMS        *pRasDialParams
  PVOID                 pvReserved
);
```

Parameters

hPort

Handle to the port on which the connection is established. Use this handle when sending or receiving data on the port.

lpszPhonebook

Pointer to a Unicode string containing the path to the phone book in which the entry for the connection resides.

lpszEntryName

Pointer to a Unicode string containing the name of the entry that was dialed to establish the connection.

pfnRasGetBuffer

Pointer to a function of type **PFNRASGETBUFFER**. The custom-scripting DLL should use this function to allocate memory to send data to the server.

pfnRasFreeBuffer

Pointer to a function of type **PFNRASFREEBUFFER**. The custom-scripting DLL should use this function to free memory allocated by the *pfnRasGetBuffer* function.

pfnRasSendBuffer

Pointer to a function of type **PFNRASSENDBUFFER**. The custom-scripting DLL uses this function to communicate with the server over the specified port.

pfnRasReceiveBuffer

Pointer to a function of type **PFNRASRECEIVEBUFFER**. The custom-scripting DLL uses this function to communicate with the server over the specified port.

pfnRasRetrieveBuffer

Pointer to a function of type **PFNRASRETRIEVEBUFFER**. The custom-scripting DLL uses this function to communicate with the server over the specified port.

hWnd

Handle to a window that the custom-scripting DLL can use to present a user interface to the user.

pRasDialParams

Pointer to a Unicode **RASDIALPARAMS** structure. This structure contains the authentication credentials for the user. The custom-scripting DLL can modify the **szUserName**, **szPassword**, and **szDomain** members of this structure. The Point-to-Point Protocol (PPP) will use whatever is stored in these members when **RasCustomScriptExecute** returns.

pvReserved

This parameter is reserved for future use.

Return Values

If the function succeeds, the return value should be ERROR_SUCCESS.

If the function fails, the return value should be an appropriate error code from Winerror.h or Raserror.h.

Remarks

When RAS calls **RasCustomScriptExecute**, the *pRasDialParams* parameter will point to a Unicode **RASDIALPARAMS** structure. That is, the structure contains only Unicode strings.

In some cases, the **szUserName** of the **RASDIALPARAMS** structure will be an empty string. In these case, the custom-scripting DLL should use the Unicode version of the **GetUserName** function to obtain the name of the current user.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Unicode: Declared only as Unicode.

See Also

RAS Custom-Scripting, **RasGetBuffer**, **RasFreeBuffer**, **RasSendBuffer**, **RasReceiveBuffer**, **RasRetrieveBuffer**

RasGetBuffer

The custom-scripting DLL calls **RasGetBuffer** to allocate memory for sending or receiving data over the port connected to the server.

```
typedef DWORD (APIENTRY *PFNRASGETBUFFER) (
  PBYTE *   ppBuffer,
  PDWORD    pdwSize
);
```

Parameters

ppBuffer

Pointer to a pointer that receives the address of the returned buffer.

pdwSize

Pointer to a **DWORD** variable that, on input, contains the requested size of the buffer. On output, this variable contains the actual size of the buffer allocated.

Return values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_OUT_OF_BUFFERS	RAS cannot allocate anymore buffer space.

Remarks

The maximum buffer size that can be obtained from is 1500 bytes.

The custom-scripting DLL calls **RasGetBuffer** through a function pointer. The function pointer is passed to the custom-scripting DLL as a parameter when RAS calls the DLL's implementation of **RasCustomScriptExecute**.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.

See Also

RAS Custom-Scripting, **RasCustomScriptExecute**, **RasFreeBuffer**

RasFreeBuffer

The custom-scripting DLL calls **RasFreeBuffer** to release a memory buffer that was allocated by a previous call to **RasGetBuffer**.

```
typedef DWORD (APIENTRY *PFNFREEBUFFER) (
  PBYTE    pBuffer
);
```

Parameters

pBuffer

Pointer to the memory buffer to free. This memory must have been obtained by a previous call to **RasGetBuffer**.

Return values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_BUFFER_INVALID	The pointer to the buffer passed in the *pBuffer* parameter is invalid.
ERROR_INVALID_PORT_HANDLE	The handle specified by the *hPort* parameter is invalid.

Remarks

The custom-scripting DLL calls **RasFreeBuffer** through a function pointer. The function pointer is passed to the custom-scripting DLL as a parameter when RAS calls the DLL's implementation of **RasCustomScriptExecute**.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.

See Also

RAS Custom-Scripting, **RasCustomScriptExecute**, **RasGetBuffer**

RasSendBuffer

The custom-scripting DLL calls the **RasSendBuffer** function to send data to the server over the specified port.

```
typedef DWORD (APIENTRY *PFNRASSENDBUFFER) (
  HANDLE    hPort,
  PBYTE     pBuffer,
  DWORD     dwSize
);
```

Parameters

hPort

Handle to the port on which to send the data in the buffer. This handle should be the handle passed in by RAS as the first parameter of the **RasCustomScriptExecute** function.

pBuffer

Pointer to a buffer of data to send over the port specified by the *hPort* parameter. Obtain this buffer using **RasGetBuffer** function.

dwSize

Specifies the size of the data in the buffer pointed to by the *pBuffer* parameter.

Return values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_BUFFER_INVALID	The pointer to the buffer passed in the *pBuffer* parameter is invalid.
ERROR_INVALID_PORT_HANDLE	The handle specified by the *hPort* parameter is invalid.

Remarks

The custom-scripting DLL calls **RasSendBuffer** through a function pointer. The function pointer is passed to the custom-scripting DLL as a parameter when RAS calls the DLL's implementation of **RasCustomScriptExecute**.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.

See Also

RAS Custom-Scripting, **RasCustomScriptExecute**, **RasReceiveBuffer**, **RasRetrieveBuffer**

RasReceiveBuffer

The custom-scripting DLL calls the **RasReceiveBuffer** function to inform RAS that it is ready to receive data from the server over the specified port.

```
typedef DWORD (APIENTRY *PFNRASRECEIVEBUFFER) (
  HANDLE    hPort,
  PBYTE     pBuffer,
  PDWORD    pdwSize,
  DWORD     dwTimeout,
  HANDLE    hEvent
);
```

Parameters

hPort
Handle to the port on which to receive the data. This handle should be the handle passed in by RAS as the first parameter of the **RasCustomScriptExecute** function.

pBuffer
Pointer to a buffer to receive the data from the port specified by the *hPort* parameter. Obtain this buffer using **RasGetBuffer** function.

pdwSize
Pointer to a **DWORD** variable that receives the size of the data returned in the buffer pointed to by the *pBuffer* parameter.

dwTimeout
Specifies a time-out period in milliseconds after which the custom-scripting DLL will no longer wait for the data.

hEvent
Handle to an event object that RAS will signal when the received data is available.

Return values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_BUFFER_INVALID	The pointer to the buffer passed in the *pBuffer* parameter is invalid.
ERROR_INVALID_PORT_HANDLE	The handle specified by the *hPort* parameter is invalid.

Remarks

RasReceiveBuffer is an asynchronous function. **RasReceiveBuffer** returns immediately even if the data is not yet available. The custom-scripting DLL must wait on the event object specified by the *hEvent* parameter. When the data is available, RAS signals this event. The custom-scripting DLL should then call the **RasRetrieveBuffer** function to obtain the data. The custom-scripting DLL may pass the same buffer pointer in **RasRetrieveBuffer** that it passed in **RasReceiveData**.

RAS also signals the event object if, for some reason, the port is disconnected before the data is posted. In this case, **RasRetrieveBuffer** returns an error defined in Raserror.h, that indicates the cause of the failure.

The custom-scripting DLL calls **RasReceiveBuffer** through a function pointer. The function pointer is passed to the custom-scripting DLL as a parameter when RAS calls the DLL's implementation of **RasCustomScriptExecute**.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.

See Also

RAS Custom-Scripting, **RasCustomScriptExecute**, **RasSendBuffer**

RasRetrieveBuffer

The custom-scripting DLL calls the **RasRetrieveBuffer** function to obtain data received from the RAS server over the specified port. The custom-scripting DLL should call **RasRetrieveBuffer** only after RAS has signaled the event object passed in the call to **RasReceiveBuffer**.

```
typedef DWORD (APIENTRY * PRASPORTRETRIEIVEBUFFER ) (
  HPORT    hPort,
  PBYTE    pBuffer
  PDWORD   pdwSize
);
```

Parameters

hPort

Handle to the port on which to receive the data. This handle should be the handle passed in by RAS as the first parameter of the **RasCustomScriptExecute** function.

pBuffer

Pointer to a buffer to receive the data from the port specified by the *hPort* parameter. Obtain this buffer using **RasGetBuffer** function. The value of this parameter may be the same as the pointer to the buffer passed into the **RasReceiveBuffer** function.

pdwSize

Pointer to a **DWORD** variable that receives the size of the data returned in the buffer pointed to by the *pBuffer* parameter.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_BUFFER_INVALID	The pointer to the buffer passed in the *pBuffer* parameter is invalid.
ERROR_INVALID_PORT_HANDLE	The handle specified by the *hPort* parameter is invalid.

RAS signals the event object if the port gets disconnected for some reason before the data is posted. In this case, **RasRetrieveBuffer** returns an error defined in Raserror.h, that indicates the cause of the failure.

Remarks

The **RasRetrieveBuffer** function is synchronous. When it returns, the buffer pointed to by the *pBuffer* parameter contains the data received over the specified port. The custom-scripting DLL should call **RasRetrieveBuffer** only after RAS has signaled the event object that the DLL passed in the call to **RasReceiveBuffer**.

The custom-scripting DLL calls **RasRetrieveBuffer** through a function pointer. The function pointer is passed to the custom-scripting DLL as a parameter when RAS calls the DLL's implementation of **RasCustomScriptExecute**.

See Also

RAS Custom-Scripting, **RasCustomScriptExecute**, **RasReceiveBuffer**, **RasSendBuffer**

CHAPTER 8

RAS Structures

Use the following structures to implement RAS functionality:

RASADPARAMS
RASAMB
RASAUTODIALENTRY
RASCONN
RASCONNSTATUS
RASCREDENTIALS
RASCTRYINFO
RASDEVINFO
RASDIALDLG
RASDIALEXTENSIONS
RASDIALPARAMS
RASEAPINFO
RASEAPUSERIDENTITY
RASENTRY
RASENTRYDLG
RASENTRYNAME
RASIPADDR
RASMONITORDLG
RASNOUSER
RASPBDLG
RASPPPCCP
RASPPPIP
RASPPPIPX
RASPPPLCP
RASPPPNBF
RASSLIP
RASSUBENTRY

RASADPARAMS

The **RASADPARAMS** structure describes the parameters that AutoDial passes to a **RASADFunc** AutoDial handler.

```
typedef struct tagRASADPARAMS {
  DWORD       dwSize;
  HWND        hwndOwner;
  DWORD       dwFlags;
  LONG        xDlg;
  LONG        yDlg;
} RASADPARAMS;
```

Members

dwSize
Specifies the size, in bytes, of the **RASADPARAMS** structure. The system sets **dwSize** to `sizeof(RASADPARAMS)` to identify the version of the structure.

hwndOwner
Specifies the parent window for the AutoDial user interface. This member can be NULL.

dwFlags

Specifies a flag that indicates how to position the window of your AutoDial user interface. The following flag is defined.

Flag	Description
RASADFLG_PositionDlg	If this flag is set, position your window according to the coordinates specified by the **xDlg** and **yDlg** members. If this flag is not set, center your window on the window specified by the **hwndOwner** member. If **hwndOwner** is NULL, center your window on the screen.

xDlg

Specifies the horizontal screen coordinate of your window's upper-left corner. Ignore this member if the RASADFLG_PositionDlg bit is not set in the **dwFlags** member.

yDlg

Specifies the vertical screen coordinate of your window's upper-left corner. Ignore this member if the RASADFLG_PositionDlg bit is not set in the **dwFlags** member.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RASADFunc**

RASAMB

The **RASAMB** structure contains the result of a Remote Access Server (RAS) Authentication Message Block (AMB) projection operation.

The **RasGetProjectionInfo** function returns a **RASAMB** data structure when its *rasprojection* parameter has the value RASP_Amb.

```
typedef struct _RASAMB {
  DWORD    dwSize;
  DWORD    dwError;
  TCHAR    szNetBiosError[ NETBIOS_NAME_LEN + 1 ];
  BYTE     bLana;
} RASAMB;
```

Members

dwSize

Specifies the size of the structure, in bytes. Before calling the **RasGetProjectionInfo** function, set this member to **sizeof**(RASAMB). The function can then determine the version of the **RASAMB** data structure that the caller of **RasGetProjectionInfo** is expecting. This allows backward compatibility for compiled applications if there are future enhancements to the data structure.

dwError

Contains the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation, the error that prevented the projection from completing successfully.

szNetBiosError

If **dwError** has the value ERROR_NAME_EXISTS_ON_NET, the **szNetBiosError** field contains a zero-terminated string that is the NetBIOS name that caused the conflict. For other values of **dwError**, this field contains the null string.

bLana

Specifies the NetBIOS network adapter identifier, or LANA, on which the remote access connection was established. This member contains the value 0xFF if a connection was not established.

Remarks

The AMB protocol is used with servers that were released before PPP was adopted as the primary framing protocol; for example, Windows NT 3.1 and OS/2 1.3 RAS servers.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetProjectionInfo**, **RASPROJECTION**

RASAUTODIALENTRY

The **RASAUTODIALENTRY** structure describes an AutoDial entry associated with a network address in the AutoDial mapping database. An AutoDial entry specifies a phone-book entry that AutoDial dials in a particular TAPI dialing location.

The **RasGetAutodialAddress** and **RasSetAutodialAddress** functions use this structure to set and retrieve information about an AutoDial entry.

```
typedef struct {
  DWORD    dwSize;
  DWORD    dwFlags;
  DWORD    dwDialingLocation;
  TCHAR    szEntry[RAS_MaxEntryName + 1];
} RASAUTODIALENTRY;
```

Members

dwSize

Specifies the size, in bytes, of the **RASAUTODIALENTRY** structure. Before calling **RasGetAutodialAddress** or **RasSetAutodialAddress**, set **dwSize** to `sizeof(RASAUTODIALENTRY)` to identify the version of the structure.

dwFlags

Reserved; must be zero.

dwDialingLocation

Specifies a TAPI dialing location. For more information about TAPI dialing locations, see the TAPI Programmer's Reference in the Platform SDK.

szEntry

Specifies a null-terminated string that names an existing phone-book entry.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetAutodialAddress**, **RasSetAutodialAddress**

RASCONN

The **RASCONN** structure provides information about a remote access connection. The **RasEnumConnections** function returns an array of **RASCONN** structures.

```
typedef struct _RASCONN {
  DWORD     dwSize;
  HRASCONN  hrasconn;
```

```
  TCHAR     szEntryName[RAS_MaxEntryName + 1];

#if (WINVER >= 0x400)
  TCHAR    szDeviceType[ RAS_MaxDeviceType + 1 ];
  TCHAR    szDeviceName[ RAS_MaxDeviceName + 1 ];
#endif
#if (WINVER >= 0x401)
  TCHAR    szPhonebook [ MAX_PATH ];
  DWORD    dwSubEntry;
#endif
#if (WINVER >= 0x500)
  GUID     guidEntry;
#endif
} RASCONN ;
```

Members

dwSize

Specifies the size, in bytes, of the **RASCONN** structure.

hrasconn

Specifies the remote access connection. This handle is used in other remote access API calls.

szEntryName

A string that specifies the phone-book entry used to establish the remote access connection. If the connection was established using an empty entry name, this string consists of a PERIOD followed by the connection phone number.

szDeviceType

Windows NT 4.0 and later: A null-terminated string that contains the device type through which the connection is made.

szDeviceName

Windows NT 4.0 and later: A null-terminated string that contains the device name through which the connection is made.

szPhonebook [MAX_PATH]

Windows NT 4.0 and later: The full path and file name to the phone book containing the entry for this connection.

dwSubEntry

Windows NT 4.0 and later: For multilink connections, specifies the subentry index of one of the connected links. Subentry indices are one based.

guidEntry

Windows 2000: A GUID (Globally Unique IDentifier) that represents the phone-book entry. The value of this member corresponds to that of the **guidid** member in the **RASENTRY** structure.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasEnumConnections**, **RasGetConnectStatus**

RASCONNSTATUS

A **RASCONNSTATUS** structure describes the current status of a remote access connection. It is returned by the **RasGetConnectStatus** function.

```
typedef struct _RASCONNSTATUS {
  DWORD         dwSize;
  RASCONNSTATE  rasconnstate;
  DWORD         dwError;
  TCHAR         szDeviceType[RAS_MaxDeviceType + 1];
  TCHAR         szDeviceName[RAS_MaxDeviceName + 1];
} RASCONNSTATUS;
```

Members

dwSize

Specifies the structure size, in bytes.

rasconnstate

Specifies a **RASCONNSTATE** enumerator value that indicates the current state of the **RasDial** connection process; that is, the piece of the **RasDial** process that is currently executing.

Two state values are especially significant.

State	Meaning
RASCS_Connected	Indicates that the connection has been successfully established.
RASCS_Disconnected	Indicates that the connection has failed.

dwError

If nonzero, indicates the reason for failure. The value is one of the error values from the RAS header file or one of ERROR_NOT_ENOUGH_MEMORY or ERROR_INVALID_HANDLE.

szDeviceType

A string that specifies the type of the current device, if available. For example, common device types supported by RAS are "modem", "pad", "switch", "isdn", or "null".

szDeviceName

A string that specifies the name of the current device, if available. This would be the name of the modem—for example, "Hayes Smartmodem 2400"; the name of the PAD, for example "US Sprint"; or the name of a switch device, for example "Racal-Guardata".

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetConnectStatus**, **RasDial**, **RASCONNSTATE**

RASCREDENTIALS

The **RASCREDENTIALS** structure is used with the **RasGetCredentials** and **RasSetCredentials** functions to specify the user credentials associated with a RAS phone-book entry.

```
typedef struct {
  DWORD dwSize;
  DWORD dwMask;
  TCHAR szUserName[UNLEN + 1];
  TCHAR szPassword[PWLEN + 1];
  TCHAR szDomain[DNLEN + 1];
} RASCREDENTIALS, *LPRASCREDENTIALS;
```

Members

dwSize

Specifies the size, in bytes, of the **RASCREDENTIALS** structure.

dwMask

Specifies a set of bit flags that specify the members of this structure that are valid. On input, set the flags to indicate the members in which you are interested. On output, the function sets the flags to indicate the members that contain valid data. This member can be a combination of the following values.

Value	Meaning
RASCM_UserName	The **szUserName** member is valid.
RASCM_Password	The **szPassword** member is valid.
RASCM_Domain	The **szDomain** member is valid.

Windows 2000 and later versions: When retrieving credentials using the **RasGetCredentials** function, the **dwMask** member contains the RASCM_Password flag if the system has saved a password for the specified entry. If the system has no password saved for this entry, **dwMask** does not contain RASCM_Password.

szUserName
Specifies a null-terminated string that contains a user name.

szPassword
Specifies a null-terminated string that contains a password.

Windows 2000 and later versions: When retrieving credentials using the **RasGetCredentials** function, the **szPassword** member does not receive the actual password. Instead, **szPassword** receives a handle to the saved password. You can substitute this handle for the saved password in calls to **RasSetCredentials** and **RasDial**. When presented with this handle, **RasDial** will retrieve and use the saved password. The value of this handle may change in future versions of the operating system; do not develop code that depends on the contents or format of this value.

szDomain
A null-terminated string that contains a domain name.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetCredentials**, **RasSetCredentials**

RASCTRYINFO

The **RASCTRYINFO** structure describes the direct dialing procedures for calls placed within a specified country. The **RasGetCountryInfo** function uses this structure to retrieve country-specific dialing information from the Windows Telephony list of country information.

For more information about country-specific dialing information, see the TAPI Programmer's Reference in the Platform SDK.

```
typedef struct RASCTRYINFO {
  DWORD   dwSize;
  DWORD   dwCountryID;
  DWORD   dwNextCountryID;
  DWORD   dwCountryCode;
  DWORD   dwCountryNameOffset;
} RASCTRYINFO;
```

Members

dwSize

Specifies the size, in bytes, of the **RASCTRYINFO** structure. Before calling **RasGetCountryInfo**, set **dwSize** to `sizeof(RASCTRYINFO)` to identify the version of the structure.

dwCountryID

Specifies a TAPI country identifier. Before calling **RasGetCountryInfo**, set **dwCountryID** to identify the country of interest. For more information about TAPI country identifiers, see the TAPI Programmer's Reference in the Platform SDK.

If this member is 1, **RasGetCountryInfo** returns information about the first country in the Windows Telephony list of country information.

dwNextCountryID

Specifies the TAPI country identifier of the next country to enumerate in the Windows Telephony list. This member is zero for the last country in the list.

dwCountryCode

Specifies the country code for the country identified by the **dwCountryID** member.

dwCountryNameOffset

Specifies the offset, in bytes, from the start of the structure to the start of a null-terminated string describing the country. The description string is either ANSI or Unicode, depending on whether you use the ANSI or Unicode version of **RasGetCountryInfo**.

Remarks

For more information on dialing procedures and telephony configuration, see the TAPI Programmer's Reference in the Platform SDK.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetCountryInfo**

RASDEVINFO

The **RASDEVINFO** structure contains information that describes a TAPI device capable of establishing a RAS connection. The **RasEnumDevices** function uses this structure to retrieve information about RAS-capable devices.

```
typedef struct tagRASDEVINFO {
  DWORD    dwSize;
  TCHAR    szDeviceType[ RAS_MaxDeviceType + 1 ];
  TCHAR    szDeviceName[ RAS_MaxDeviceName + 1 ];
} RASDEVINFO;
```

Members

dwSize

Specifies the size, in bytes, of the **RASDEVINFO** structure. Before calling **RasEnumDevices**, set **dwSize** to `sizeof(RASDEVINFO)` to identify the version of the structure.

szDeviceType

Specifies a null-terminated string indicating the RAS device type referenced by **szDeviceName**. This member can be one of the following string constants.

String	Description
RASDT_Modem	A modem accessed through a COM port.
RASDT_Isdn	An ISDN card with the corresponding NDISWAN driver installed.
RASDT_X25	An X.25 card with the corresponding NDISWAN driver installed.
RASDT_Vpn	A virtual private network connection.
RASDT_Pad	A Packet Assembler/Disassembler.

Windows 95: The RASDT_Vpn device type is supported on Windows 95 only if Microsoft Dial-Up Networking Version 1.2 is installed. The RASDT_X25 and RASDT_Pad device types are not supported on Windows 95.

Windows 98: The RASDT_Vpn device type is supported on Windows 98. However, the RASDT_X25 and RASDT_Pad device types are not currently supported on Windows 98

szDeviceName

Specifies a null-terminated string containing the name of a TAPI device.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasEnumDevices**

RASDIALDLG

The **RASDIALDLG** structure is used in the **RasDialDlg** function to specify additional input and output parameters.

```
typedef struct tagRASDIALDLG {
  IN  DWORD          dwSize;
  IN  HWND           hwndOwner;
  IN  DWORD          dwFlags;
  IN  LONG           xDlg;
  IN  LONG           yDlg;
  IN  DWORD          dwSubEntry;
  OUT DWORD          dwError;
  IN  DWORD          reserved;
  IN  DWORD          reserved2;
} RASDIALDLG;
```

Members

dwSize

Specifies the size of this structure, in bytes. Before calling **RasDialDlg**, set this member to **sizeof(RASDIALDLG)** to indicate the version of the structure. If **dwSize** is not a valid size, **RasDialDlg** fails and sets the **dwError** member to ERROR_INVALID_SIZE.

hwndOwner

Specifies the window that owns the modal **RasDialDlg** dialog boxes. This member can be any valid window handle, or it can be NULL if the dialog box has no owner.

dwFlags

A bit flag that indicates the options that are enabled for the dialog box. You can specify the following value.

Value	Meaning
RASDDFLAG_PositionDlg	If this flag is set, **RasDialDlg** uses the values specified by the **xDlg** and **yDlg** members to position the dialog box. If this flag is not set, the dialog box is centered on the owner window, unless **hwndOwner** is NULL, in which case, the dialog box is centered on the screen.

xDlg
Specifies the horizontal screen coordinate of the upper-left corner of the dialog box. This value is used only if the RASDDFLAG_PositionDlg flag is set.

yDlg
Specifies the vertical screen coordinate of the upper-left corner of the dialog box. This value is used only if the RASDDFLAG_PositionDlg flag is set.

dwSubEntry
Specifies the subentry or subentries to dial. If **dwSubEntry** is zero, **RasDialDlg** dials all subentries associated with the specified phone-book entry. Otherwise, to indicate the index of the individual subentry to dial, **dwSubEntry** must be a number from one to the number of subentries.

dwError
The **RasDialDlg** function sets this member to a system error code or RAS error code if an error occurs. If no error occurs, the function sets **dwError** to zero. This value is ignored on input.

reserved
Reserved; must be zero.

reserved2
Reserved; must be zero.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasDialDlg**

RASDIALEXTENSIONS

The **RASDIALEXTENSIONS** structure contains information about extended features of the **RasDial** function. You can enable one or more of these extensions by passing a pointer to a **RASDIALEXTENSIONS** structure when you call **RasDial**. If you do not pass a pointer to a **RASDIALEXTENSIONS** structure to **RasDial**, **RasDial** uses the default settings that are noted in the following descriptions.

```
typedef struct tagRASDIALEXTENSIONS {
  DWORD     dwSize;
  DWORD     dwfOptions;
  HWND      hwndParent;
  ULONG_PTR reserved;
#if (WINVER >= 0x500)
  ULONG_PTR reserved1;
  RASEAPINFO RasEapInfo;
#endif
} RASDIALEXTENSIONS;
```

Members

dwSize

Specifies the size of this structure, in bytes. Set this member to **sizeof(RASDIALEXTENSIONS)**. This indicates the version of the structure.

dwfOptions

A set of bit flags that specify **RasDial** extensions. The following bit flags are defined; you must set all undefined bits to zero.

Value	Description
RDEOPT_ UsePrefixSuffix	If this bit flag is one, **RasDial** uses the prefix and suffix that is in the RAS phone book. If this bit flag is zero, **RasDial** ignores the prefix and suffix that is in the RAS phone book. If no phone-book entry name is specified in the call to **RasDial**, the actual value of this bit flag is ignored, and it is assumed to be zero.
RDEOPT_ PausedStates	If this bit flag is one, **RasDial** accepts paused states. Examples of paused states are terminal mode, retry logon, change password, set callback number, and EAP authentication. If this bit flag is zero, **RasDial** reports a fatal error if it enters a paused state.

(continued)

(continued)

Value	Description
RDEOPT_ IgnoreModemSpeaker	If this bit flag is one, **RasDial** ignores the modem speaker setting that is in the RAS phone book, and uses the setting specified by the RDEOPT_SetModemSpeaker bit flag. If this bit flag is zero, **RasDial** uses the modem speaker setting that is in the RAS phone book, and ignores the setting specified by the RDEOPT_SetModemSpeaker bit flag. If no phone-book entry name is specified in the call to **RasDial**, the choice is between using a default setting or the setting specified by the RDEOPT_SetModemSpeaker bit flag. The default setting is used if RDEOPT_IgnoreModemSpeaker is zero. The setting specified by RDEOPT_SetModemSpeaker is used if RDEOPT_IgnoreModemSpeaker is one.
RDEOPT_ SetModemSpeaker	If this bit flag is one, and RDEOPT_IgnoreModemSpeaker is one, **RasDial** sets the modem speaker on. If this bit flag is zero, and RDEOPT_IgnoreModemSpeaker is one, **RasDial** sets the modem speaker off. If RDEOPT_IgnoreModemSpeaker is zero, **RasDial** ignores the value of RDEOPT_SetModemSpeaker, and sets the modem speaker based on the RAS phone-book setting or the default setting.
RDEOPT_ IgnoreSoftwareCompression	If this bit flag is one, **RasDial** ignores the software compression setting that is in the RAS phone book, and uses the setting specified by the RDEOPT_SetSoftwareCompression bit flag. If this bit flag is zero, **RasDial** uses the software compression setting that is in the RAS phone book, and ignores the setting specified by the RDEOPT_SetSoftwareCompression bit flag. If no phone-book entry name is specified in the call to **RasDial**, the choice is between using a default setting or the setting specified by the RDEOPT_SetSoftwareCompression bit flag. The default setting is used if RDEOPT_IgnoreSoftwareCompression is zero. The setting specified by RDEOPT_SetSoftwareCompression is used if RDEOPT_IgnoreSoftwareCompression is one.
RDEOPT_ SetSoftwareCompression	If this bit flag is one, and RDEOPT_IgnoreSoftwareCompression is one, **RasDial** uses software compression. If this bit flag is zero, and RDEOPT_IgnoreSoftwareCompression is one, **RasDial** does not use software compression. If RDEOPT_IgnoreSoftwareCompression is zero, **RasDial** ignores the value of RDEOPT_SetSoftwareCompression, and sets the software compression state based on the RAS phone-book setting or the default setting.

Value	Description
RDEOPT_ PauseOnScript	Used internally by the **RasDialDlg** function so that a Windows-95-style logon script is executed in a terminal window visible to the user. Applications should not set this flag.

The default value for each of these bit flags is zero.

hwndParent
Handle to a parent window that a security DLL can use for dialog box creation and centering.

Note that this is not the window that receives **RasDial** progress notifications.

This member is optional; it is not required when no security DLL is defined.

The default value for this member is NULL.

reserved
This member is reserved for future use. It must be set to zero.

reserved1
Windows 2000: This member is reserved for future use. It must be set to zero.

RasEapInfo
Windows 2000: A **RASEAPINFO** structure that contains user-specific Extensible Authentication Protocol (EAP) information.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasDial**, **RasInvokeEapUI**

RASDIALPARAMS

The **RASDIALPARAMS** structure contains parameters that are used by **RasDial** to establish a remote access connection.

```
typedef struct _RASDIALPARAMS {
  DWORD  dwSize;
  TCHAR  szEntryName[RAS_MaxEntryName + 1];
  TCHAR  szPhoneNumber[RAS_MaxPhoneNumber + 1];
  TCHAR  szCallbackNumber[RAS_MaxCallbackNumber + 1];
  TCHAR  szUserName[UNLEN + 1];
```

(continued)

(continued)

```
  TCHAR  szPassword[PWLEN + 1];
  TCHAR  szDomain[DNLEN + 1] ;
#if (WINVER >= 0x401)
  DWORD      dwSubEntry;
  ULONG_PTR  dwCallbackId;
#endif
} RASDIALPARAMS;
```

Members

dwSize

Specifies the structure size, in bytes.

szEntryName

Specifies a string containing the phone-book entry to use to establish the connection. An empty string ("") specifies a simple modem connection on the first available modem port, in which case a nonempty **szPhoneNumber** must be provided.

Windows NT 4.0 and later versions: The callback number is no longer stored in the registry. Specifying an asterisk for **szCallbackNumber** causes RAS to return error 704: ERROR_BAD_CALLBACK_NUMBER.

szPhoneNumber

Specifies a string that contains an overriding phone number. An empty string ("") indicates that the phone-book entry's phone number should be used. If **szEntryName** is "", **szPhoneNumber** cannot be "".

szCallbackNumber

Specifies a string that contains a callback phone number. An empty string ("") indicates that callback should not be used. This string is ignored unless the user has "Set By Caller" callback permission on the RAS server. An asterisk indicates that the number stored in the phone book should be used for callback.

szUserName

Specifies a string that contains the user's user name. This string is used to authenticate the user's access to the remote access server.

szPassword

Specifies a string that contains the user's password. This string is used to authenticate the user's access to the remote access server.

Windows NT/2000: You can use **szPassword** to send a new password to the remote server when you restart a **RasDial** connection from a RASCS_PasswordExpired paused state. When changing a password on an entry that calls Microsoft Networks, you should limit the new password to 14 characters in length to avoid down-level compatibility problems.

Windows 2000 and later versions: When retrieving the password using the **RasGetEntryDialParams** function, the **szPassword** member does not receive the actual password. Instead, **szPassword** receives a handle to the saved password. You can substitute this handle for the saved password in calls to **RasSetDialParams**, and **RasDial**. When presented with this handle, **RasDial** retrieves and uses the saved password. The value of this handle may change in future versions of the operating system; do not develop code that depends on the contents or format of this value.

szDomain

Specifies a string that contains the domain on which authentication is to occur. An empty string ("") specifies the domain in which the remote access server is a member. An asterisk specifies the domain stored in the phone book for the entry.

dwSubEntry

Specifies the index of the initial subentry to dial. If the dial mode is RASEDM_DialAsNeeded, RAS dials this subentry. If **dwSubEntry** is not a valid subentry index, RAS dials the first subentry.

If the dial mode of the phone-book entry is RASEDM_DialAll, **dwSubEntry** is ignored. If the phone-book entry has no subentries, **dwSubEntry** is ignored.

The subentry indices are one-based. That is, the first subentry has an index of one, the second subentry as an index of two, and so on.

The **RASENTRY** structure returned by **RasGetEntryProperties** indicates the dial mode (**dwDialMode**) and number of subentries (**dwSubEntries**) for the phone-book entry.

Windows 2000 and later: If **dwSubEntry** specifies a valid subentry index, RAS dials the specified subentry regardless of the dial mode. If the dial mode is RASEDM_DialAll and **dwSubEntry** is zero, RAS dials all of the subentries.

dwCallbackId

Specifies an application-defined value that RAS passes to your **RasDialFunc2** callback function.

Remarks

The **szUserName** and **szPassword** strings are used to authenticate the user's access to the remote access server.

Windows NT/2000: RAS does not actually log the user onto the network. The user does this in the usual manner, for example, by logging on with cached credentials prior to making the connection, or by using CTRL+ALT+DEL after the RAS connection is established.

If both the **szUserName** and **szPassword** members are empty strings (""), RAS uses the user name and password of the current logon context for authentication. For a user-mode application, RAS uses the credentials of the currently logged-on interactive user. For a Win32 service process, RAS uses the credentials associated with the service.

Windows 95: RAS uses the **szUserName** and **szPassword** strings to log the user onto the network.

Windows 95 cannot obtain the password of the currently logged-on user, so if both the **szUserName** and the **szPassword** members are empty strings (""), RAS leaves the user name and password empty during authentication.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasDial**, **RasGetEntryProperties**, **RasSetEntryDialParams**, **RASENTRY**

RASEAPINFO

The **RASEAPINFO** structure contains user-specific Extensible Authentication Protocol (EAP) information. Use **RASEAPINFO** to pass this information to the **RasDial** function.

```
typedef struct tagRASEAPINFO {
  DWORD    dwSizeofEapInfo;
  BYTE   *  pbEapInfo;
};
```

Members

dwSizeofEapInfo
Specifies the size of the binary information pointed to by the **pbEapInfo** member.

pbEapInfo
Pointer to binary EAP information. **RasDial** uses this information for authentication.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetEapUserData**, **RASDIALEXTENSIONS**

RASEAPUSERIDENTITY

The **RASEAPUSERIDENTITY** structure stores identity information for a particular user. This information is required for remote access connections that use Extensible Authentication Protocol (EAP) for authentication.

```
struct RASEAPUSERIDENTITY {
  TCHAR    szUserName[ UNLEN + 1 ]; // user name
  DWORD    dwSizeofEapInfo;         // size of identity info
  BYTE     pbEapInfo[ 1 ];          // identity info
};
```

Members

szUserName[UNLEN + 1]
Pointer to user name of the user requesting authentication.

dwSizeofEapInfo
Size of the identity information required by the extensible authentication protocol.

pbEapInfo[1]
Pointer to the identity information required by the extensible authentication protocol.

Remarks

Obtain the EAP information for the current user by calling **RasGetEapUserIdentity**. This function will return a **RASEAPUSERIDENTITY** structure containing the EAP information. Free the memory occupied by this structure by calling **RasFreeEapUserIdentity.**

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

RasFreeEapUserIdentity, **RasGetEapUserIdentity**

RASENTRY

The **RASENTRY** structure describes a phone-book entry. The **RasSetEntryProperties** and **RasGetEntryProperties** functions use this structure to set and retrieve the properties of a phone-book entry.

```
typedef struct tagRASENTRY {
  DWORD      dwSize;
  DWORD      dwfOptions;
  //
  // Location/phone number.
  //
  DWORD      dwCountryID;
  DWORD      dwCountryCode;
```

(continued)

(continued)

```
TCHAR       szAreaCode[ RAS_MaxAreaCode + 1 ];
TCHAR       szLocalPhoneNumber[ RAS_MaxPhoneNumber + 1 ];
DWORD       dwAlternateOffset;
//
// PPP/Ip
//
RASIPADDR   ipaddr;
RASIPADDR   ipaddrDns;
RASIPADDR   ipaddrDnsAlt;
RASIPADDR   ipaddrWins;
RASIPADDR   ipaddrWinsAlt;
//
// Framing
//
DWORD       dwFrameSize;
DWORD       dwfNetProtocols;
DWORD       dwFramingProtocol;
//
// Scripting
//
TCHAR       szScript[ MAX_PATH ];
//
// AutoDial
//
TCHAR        szAutodialDll[ MAX_PATH ];
TCHAR        szAutodialFunc[ MAX_PATH ];
//
// Device
//
TCHAR       szDeviceType[ RAS_MaxDeviceType + 1 ];
TCHAR       szDeviceName[ RAS_MaxDeviceName + 1 ];
//
// X.25
//
TCHAR       szX25PadType[ RAS_MaxPadType + 1 ];
TCHAR       szX25Address[ RAS_MaxX25Address + 1 ];
TCHAR       szX25Facilities[ RAS_MaxFacilities + 1 ];
TCHAR       szX25UserData[ RAS_MaxUserData + 1 ];
DWORD       dwChannels;
//
// Reserved
//
DWORD        dwReserved1;
DWORD        dwReserved2;
```

```
#if (WINVER >= 0x401)
    //
    // Multilink and BAP
    //
    DWORD       dwSubEntries;
    DWORD       dwDialMode;
    DWORD       dwDialExtraPercent;
    DWORD       dwDialExtraSampleSeconds;
    DWORD       dwHangUpExtraPercent;
    DWORD       dwHangUpExtraSampleSeconds;
    //
    // Idle time out
    //
    DWORD       dwIdleDisconnectSeconds;
#endif
#if (WINVER >= 0x500)
    DWORD     dwType;             // entry type
    DWORD     dwEncryptionType;   // type of encryption to use
    DWORD     dwCustomAuthKey;    // authentication key for EAP
    GUID      guidId;             // guid that represents
                                  // the phone-book entry
    TCHAR    szCustomDialDll[MAX_PATH];   // DLL for custom
                                          // dialing
    DWORD    dwVpnStrategy;       // specifies type of VPN
                                  // protocol
#endif

} RASENTRY;
```

Members

dwSize

Specifies the size, in bytes, of the **RASENTRY** structure. Before calling **RasSetEntryProperties** or **RasGetEntryProperties**, set **dwSize** to `sizeof(RASENTRY)` to identify the version of the structure.

dwfOptions

A set of bit flags that specify connection options. You can set one or more of the following flags.

Flag	Description
RASEO_ UseCountryAndAreaCodes	If this flag is set, the **dwCountryID**, **dwCountryCode**, and **szAreaCode** members are used to construct the phone number. If this flag is not set, these members are ignored. This flag corresponds to the Use Country and Area Codes check boxes in the Phone dialog box.

(continued)

(continued)

Flag	Description
RASEO_ SpecificIpAddr	If this flag is set, RAS tries to use the IP address specified by **ipaddr** as the IP address for the dial-up connection. If this flag is not set, the value of the **ipaddr** member is ignored. Setting the RASEO_SpecificIpAddr flag corresponds to selecting the Specify an IP Address setting in the TCP/IP settings dialog box. Clearing the RASEO_SpecificIpAddr flag corresponds to selecting the Server Assigned IP Address setting in the TCP/IP settings dialog box. Currently, an IP address set in the phone-book entry properties or retrieved from a server overrides the IP address set in the network control panel.
RASEO_ SpecificNameServers	If this flag is set, RAS uses the **ipaddrDns, ipaddrDnsAlt, ipaddrWins**, and **ipaddrWinsAlt** members to specify the name server addresses for the dial-up connection. If this flag is not set, RAS ignores these members. Setting the RASEO_SpecificNameServers flag corresponds to selecting the Specify Name Server Addresses setting in the TCP/IP Settings dialog box. Clearing the RASEO_SpecificNameServers flag corresponds to selecting the Server Assigned Name Server Addresses setting in the TCP/IP Settings dialog box.
RASEO_ IpHeaderCompression	If this flag is set, RAS negotiates to use IP header compression on PPP connections. If this flag is not set, IP header compression is not negotiated. This flag corresponds to the Use IP Header Compression check box in the TCP/IP settings dialog box. It is generally advisable to set this flag because IP header compression significantly improves performance. The flag should be cleared only when connecting to a server that does not correctly negotiate IP header compression.
RASEO_ RemoteDefaultGateway	If this flag is set, the default route for IP packets is through the dial-up adapter when the connection is active. If this flag is clear, the default route is not modified. This flag corresponds to the Use Default Gateway on Remote Network check box in the TCP/IP settings dialog box.
RASEO_ DisableLcpExtensions	If this flag is set, RAS disables the PPP LCP extensions defined in RFC 1570. This may be necessary to connect to certain older PPP implementations, but interferes with features such as server callback. Do not set this flag unless specifically required.
RASEO_ TerminalBeforeDial	If this flag is set, RAS displays a terminal window for user input before dialing the connection.

Flag	Description
RASEO_ TerminalAfterDial	If this flag is set, RAS displays a terminal window for user input after dialing the connection. Do not set this flag if a dial-up networking script is to be associated with the connection, because scripting has its own terminal implementation.
RASEO_ ModemLights	**Windows 2000:** If this flag is set, a status monitor will be displayed in the Task Bar.
RASEO_ SwCompression	If this flag is set, software compression is negotiated on the link. Setting this flag causes the PPP driver to attempt to negotiate CCP with the server. This flag should be set by default, but clearing it can reduce the negotiation period if the server does not support a compatible compression protocol.
RASEO_ RequireEncryptedPw	If this flag is set, only secure password schemes can be used to authenticate the client with the server. This prevents the PPP driver from using the PAP plain-text authentication protocol to authenticate the client. The CHAP and SPAP authentication protocols are also supported. Clear this flag for increased interoperability, and set it for increased security. This flag corresponds to the Require Encrypted Password check box in the Security dialog box. See also RASEO_RequireMsEncryptedPw.
RASEO_ RequireMsEncryptedPw	If this flag is set, only the Microsoft secure password schemes can be used to authenticate the client with the server. This prevents the PPP driver from using the PPP plain-text authentication protocol, MD5-CHAP, MS-CHAP, or SPAP. The flag should be cleared for maximum interoperability and should be set for maximum security. This flag takes precedence over RASEO_RequireEncryptedPw. This flag corresponds to the Require Microsoft Encrypted Password check box in the Security dialog box. See also RASEO_RequireDataEncryption.
RASEO_ RequireDataEncryption	If this flag is set, data encryption must be negotiated successfully or the connection should be dropped. This flag is ignored unless RASEO_RequireMsEncryptedPw is also set. This flag corresponds to the Require Data Encryption check box in the Security dialog box.
RASEO_ NetworkLogon	If this flag is set, RAS logs on to the network after the point-to-point connection is established. This flag currently has no effect under Windows NT/2000.

(continued)

(continued)

Flag	Description
RASEO_ UseLogonCredentials	If this flag is set, RAS uses the user name, password, and domain of the currently logged-on user when dialing this entry. This flag is ignored unless RASEO_RequireMsEncryptedPw is also set. Note that this setting is ignored by the **RasDial** function, where specifying empty strings for the **szUserName** and **szPassword** members of the **RASDIALPARAMS** structure gives the same result. This flag corresponds to the Use Current Username and Password check box in the Security dialog box.
RASEO_ PromoteAlternates	This flag has an effect when alternate phone numbers are defined by the **dwAlternateOffset** member. If this flag is set, an alternate phone number that connects successfully becomes the primary phone number, and the current primary phone number is moved to the alternate list. This flag corresponds to the check box in the Alternate Numbers dialog box.
RASEO_ SecureLocalFiles	**Windows NT/2000:** If this flag is set, RAS checks for existing remote file system and remote printer bindings before making a connection with this entry. Typically, you set this flag on phone-book entries for public networks to remind users to break connections to their private network before connecting to a public network.
RASEO_ RequireEAP	**Windows 2000:** If this flag is set, an Extensible Authentication Protocol (EAP) must be supported for authentication.
RASEO_ RequirePAP	**Windows 2000:** If this flag is set, Password Authentication Protocol must be supported for authentication.
RASEO_ RequireSPAP	**Windows 2000:** If this flag is set, Shiva's Password Authentication Protocol must be supported for authentication.
RASEO_ Custom	**Windows 2000:** If this flag is set, the connection will use custom encryption.
RASEO_ PreviewPhoneNumber	**Windows 2000:** If this flag is set, the remote access dialer displays the phone number to be dialed.
RASEO_ SharedPhoneNumbers	**Windows 2000:** If this flag is set, phone numbers are shared.
RASEO_ ReviewUserPW	**Windows 2000:** If this flag is set, the remote access dialer displays the user's name and password prior to dialing.
RASEO_ PreviewDomain	**Windows 2000:** If this flag is set, the remote access dialer displays the domain name prior to dialing.
RASEO_ ShowDialingProgress	**Windows 2000:** If this flag is set, the remote access dialer displays its progress in establishing the connection.

Flag	Description
RASEO_ RequireCHAP	**Windows 2000:** If this flag is set, the Challenge Handshake Authentication Protocol must be supported for authentication.
RASEO_ RequireMsCHAP	**Windows 2000:** If this flag is set, the Microsoft Challenge Handshake Authentication Protocol must be supported for authentication.
RASEO_ RequireMsCHAP2	**Windows 2000:** If this flag is set, version 2 of the Microsoft Challenge Handshake Authentication Protocol must be supported for authentication.
RASEO_ RequireW95MSCHAP	**Windows 2000:** If this flag is set, MS-CHAP must send the LanManager-hashed password.
RASEO_ CustomScript	**Windows 2000:** If this flag is set, RAS will invoke a custom-scripting DLL after establishing the connection to the server.

dwCountryID

Specifies the TAPI country identifier. Use the **RasGetCountryInfo** function to enumerate country identifiers. This member is ignored unless the **dwfOptions** member specifies the RASEO_UseCountryAndAreaCodes flag.

dwCountryCode

Specifies the country code portion of the phone number. The country code must correspond to the country identifier specified by **dwCountryID**. If **dwCountryCode** is zero, the country code is based on the country identifier specified by **dwCountryID**. This member is ignored unless **dwfOptions** specifies the RASEO_UseCountryAndAreaCodes flag.

szAreaCode

Specifies the area code as a null-terminated string. If the dialing location does not have an area code, specify an empty string (""). Do not include parentheses or other delimiters in the area code string. (For example, "206" is a valid area code; "(206)" is not. This member is ignored unless the **dwfOptions** member specifies the RASEO_UseCountryAndAreaCodes flag.

szLocalPhoneNumber

Specifies a null-terminated string containing a telephone number. The way RAS uses this string depends on whether the **dwfOptions** member specifies the RASEO_UseCountryAndAreaCodes flag. If the flag is set, RAS combines **szLocalPhoneNumber** with the country and area codes specified by the **dwCountryID**, **dwCountryCode** , and **szAreaCode** members. If the flag is not set, RAS uses the **szLocalPhoneNumber** string as the entire phone number.

dwAlternateOffset

Specifies the offset, in bytes, from the beginning of the structure to a list of consecutive null-terminated strings. The last string is terminated by two consecutive null characters. The strings are alternate phone numbers that RAS dials in the order listed if the primary number (see **szLocalPhoneNumber**) fails to connect. The alternate phone number strings are ANSI or Unicode, depending on whether you use the ANSI or Unicode version of the structure.

ipaddr
Specifies the IP address to be used while this connection is active. This member is ignored unless **dwfOptions** specifies the RASEO_SpecificIpAddr flag.

ipaddrDns
Specifies the IP address of the DNS server to be used while this connection is active. This member is ignored unless **dwfOptions** specifies the RASEO_SpecificNameServers flag.

ipaddrDnsAlt
Specifies the IP address of a secondary or backup DNS server to be used while this connection is active. This member is ignored unless **dwfOptions** specifies the RASEO_SpecificNameServers flag.

ipaddrWins
Specifies the IP address of the WINS server to be used while this connection is active. This member is ignored unless **dwfOptions** specifies the RASEO_SpecificNameServers flag.

ipaddrWinsAlt
Specifies the IP address of a secondary WINS server to be used while this connection is active. This member is ignored unless **dwfOptions** specifies the RASEO_SpecificNameServers flag.

dwFrameSize
Specifies the network protocol frame size. The value should be either 1006 or 1500. This member is ignored unless **dwFramingProtocol** specifies the RASFP_Slip flag.

dwfNetProtocols
Specifies the network protocols to negotiate. This member can be a combination of the following flags.

Flag	Description
RASNP_NetBEUI	Negotiate the NetBEUI protocol.
RASNP_Ipx	Negotiate the IPX protocol.
RASNP_Ip	Negotiate the TCP/IP protocol.

dwFramingProtocol
Specifies the framing protocol used by the server. PPP is the emerging standard. SLIP is used mainly in UNIX environments. This member can be one of the following flags.

Flag	Description
RASFP_Ppp	Point-to-Point Protocol (PPP)
RASFP_Slip	Serial Line Internet Protocol (SLIP)
RASFP_Ras	Asynchronous NetBEUI, Microsoft proprietary protocol implemented in Windows NT 3.1 and Windows for Workgroups 3.11

To use Compressed SLIP, set the RASFP_Slip flag and set the RASEO_IpHeaderCompression flag in the **dwfOptions** member.

Windows 2000 or later: The RASFP_Ras flag is no longer supported. As a result, Windows 2000 and later computers will not be able to connect to Lan Manager, Windows for Workgroups 3.11, or Windows NT 3.1 servers. However, these earlier platforms will continue to be able to connect to Windows 2000 and later servers.

szScript

Specifies a null-terminated string containing the name of the script file. The file name should be a full path.

Windows NT/2000: To indicate a Windows NT/Windows 2000 SWITCH.INF script name, set the first character of the name to "[".

szAutodialDll

Specifies a null-terminated string containing the full path and file name of the Dynamic-Link Library (DLL) for the customized AutoDial handler. If **szAutodialDll** contains an empty string (""), RAS uses the default dialing user interface and the **szAutodialFunc** member is ignored.

szAutodialFunc

Specifies a null-terminated string containing the exported name of the **RASADFunc** function for the customized AutoDial handler. An AutoDial DLL must provide both ANSI and Unicode versions of the **RASADFunc** handler. However, do not include the "A" or "W" suffix in the name specified by **szAutodialFunc**.

szDeviceType

Specifies a null-terminated string indicating the RAS device type referenced by **szDeviceName**. This member can be one of the following string constants.

String	Description
RASDT_Modem	A modem accessed through a COM port.
RASDT_Isdn	An ISDN card with corresponding NDISWAN driver installed.
RASDT_X25	An X.25 card with corresponding NDISWAN driver installed.
RASDT_Vpn	**Windows 2000:** A virtual private network connection.
RASDT_Pad	**Windows 2000:** A Packet Assembler/Disassembler.
RASDT_Generic	**Windows 2000:** Generic
RASDT_Serial	**Windows 2000:** Direct serial connection through a serial port.
RASDT_FrameRelay	**Windows 2000:** Frame Relay
RASDT_Atm	**Windows 2000:** Asynchronous Transfer Mode
RASDT_Sonet	**Windows 2000:** Sonet
RASDT_SW56	**Windows 2000:** Switched 56K Access

(continued)

(continued)

String	Description
RASDT_Irda	**Windows 2000:** Infrared Data Association (IrDA) compliant device.
RASDT_Parallel	**Windows 2000:** Direct parallel connection through a parallel port.

Windows 95: The RASDT_Vpn device type is supported on Windows 95 only if Microsoft Dial-Up Networking Version 1.2 is installed. The RASDT_X25 and RASDT_Pad device types are not supported on Windows 95.

Windows 98: The RASDT_Vpn device type is supported on Windows 98. However, the RASDT_X25 and RASDT_Pad device types are not currently supported on Windows 98.

szDeviceName

Contains a null-terminated string containing the name of a TAPI device to use with this phone-book entry, for example, "XYZ Corp 28800 External". To enumerate all available RAS-capable devices, use the **RasEnumDevices** function.

szX25PadType

Contains a null-terminated string that identifies the X.25 PAD type. Set this member to "" unless the entry should dial using an X.25 PAD.

Windows NT/2000: Under Windows NT/Windows 2000, the **szX25PadType** string maps to a section name in PAD.INF.

szX25Address

Contains a null-terminated string that identifies the X.25 address to connect to. Set this member to "" unless the entry should dial using an X.25 PAD or native X.25 device.

szX25Facilities

Contains a null-terminated string that specifies the facilities to request from the X.25 host at connection. This member is ignored if **szX25Address** is an empty string ("").

szX25UserData

Contains a null-terminated string that specifies additional connection information supplied to the X.25 host at connection. This member is ignored if **szX25Address** is an empty string ("").

dwChannels;

dwReserved1

Reserved; must be zero.

dwReserved2

Reserved; must be zero.

dwSubEntries

Specifies the number of multilink subentries associated with this entry. When calling **RasSetEntryProperties**, set this member to zero. To add subentries to a phone-book entry, use the **RasSetSubEntryProperties** function.

dwDialMode

Specifies whether RAS should dial all of this entry's multilink subentries when the entry is first connected. This member can be one of the following values.

Value	Meaning
RASEDM_DialAll	Dial all subentries initially.
RASEDM_DialAsNeeded	Adjust the number of subentries as bandwidth is needed. RAS uses the **dwDialExtraPercent, dwDialExtraSampleSeconds, dwDialHangUpExtraPercent,** and **dwHangUpExtraSampleSeconds** members to determine when to dial or disconnect a subentry.

Windows 2000 and later: In order for RAS to dial all subentries, **dwDialMode** must be set to RASEDM_DialAll and the **dwSubEntry** member of **RASDIALPARAMS** must be set to zero.

dwDialExtraPercent

Windows 2000 or later: Specifies a percent of the total bandwidth available from the currently connected subentries. RAS dials an additional subentry when the total bandwidth used exceeds **dwDialExtraPercent** percent of the available bandwidth for at least **dwDialExtraSampleSeconds** seconds.

This member is ignored unless the **dwDialMode** member specifies the RASEDM_DialAsNeeded flag.

dwDialExtraSampleSeconds

Windows 2000 or later: Specifies the number of seconds that current bandwidth usage must exceed the threshold specified by **dwDialExtraPercent** before RAS dials an additional subentry.

This member is ignored unless the **dwDialMode** member specifies the RASEDM_DialAsNeeded flag.

dwHangUpExtraPercent

Windows 2000 or later: Specifies a percent of the total bandwidth available from the currently connected subentries. RAS terminates (hangs up) an existing subentry connection when total bandwidth used is less than **dwHangUpExtraPercent** percent of the available bandwidth for at least **dwHangUpExtraSampleSeconds** seconds.

This member is ignored unless the **dwDialMode** member specifies the RASEDM_DialAsNeeded flag.

dwHangUpExtraSampleSeconds

Windows 2000 or later: Specifies the number of seconds that current bandwidth usage must be less than the threshold specified by **dwHangUpExtraPercent** before RAS terminates an existing subentry connection.

This member is ignored unless the **dwDialMode** member specifies the RASEDM_DialAsNeeded flag.

dwIdleDisconnectSeconds

Specifies the number of seconds after which the connection is terminated due to inactivity. Note that unless the idle time out is disabled, the entire connection is terminated if the connection is idle for the specified interval. This member can specify a number of seconds, or one of the following values.

Value	Meaning
RASIDS_Disabled	There is no idle time out for this connection.
RASIDS_UseGlobalValue	Use the user preference value as the default.

dwType

Windows 2000: The type of phone-book entry. This member can be one of the following types.

Type	Description
RASET_Phone	Phone line, for example, modem, ISDN, X.25.
RASET_Vpn	Virtual Private Network
RASET_Direct	Direct serial or parallel connection
RASET_Internet	Internet Connection Services (ICS)

dwEncryptionType

Windows 2000: The type of encryption to use for Microsoft Point to Point Encryption (MPPE) with the connection. This member can be one of the following values.

Value	Meaning
ET_40Bit	Require encryption
ET_128Bit	Require strong encryption
ET_None	No encryption
ET_Require	Require encryption
ET_RequireMax	Require maximum-strength encryption.
ET_Optional	Do encryption if possible. No encryption is okay.

The value of **dwEncryptionType** does not affect how passwords are encrypted. Whether passwords are encrypted and how passwords are encrypted is determined by the authentication protocol, e.g. PAP, MS-CHAP, EAP.

dwCustomAuthKey

Windows 2000: This member is used for Extensible Authentication Protocol (EAP). This member contains the authentication key provided to the EAP vendor.

guidId

Windows 2000: The GUID (Globally Unique IDentifier) that represents this phone-book entry. This member is not settable.

szCustomDialDll[MAX_PATH]

Windows 2000: A null-terminated string containing the full path and file name for the dynamic link library (DLL) that implements the custom-dialing functions. This DLL should export Unicode versions of functions named **RasCustomDial**, **RasCustomHangup**, **RasCustomEntryDlg**, and **RasCustomDialDlg**. These functions should have prototypes *RasCustomDialFn* and *RasCustomHangUpFn* as defined in Ras.h, and *RasCustomDialDlgFn* and *RasCustomEntryDlgFn* as defined in Rasdlg.h.

If **szCustomDialDll** contains an empty string, RAS uses the default system dialer.

dwVpnStrategy

Windows 2000: The VPN strategy to use when dialing a VPN connection. This member can have one of the following values.

Value	Meaning
VS_Default	With this strategy, RAS dials PPTP first. If PPTP fails, L2TP is attempted. Whichever protocol succeeds is tried first in subsequent dialing for this entry.
VS_PptpOnly	RAS will dial only PPTP.
VS_PptpFirst	RAS will always dial PPTP first.
VS_L2tpOnly	RAS will dial only L2TP.
VS_L2tpFirst	RAS will always dial L2TP first.

Remarks

Unless the operating system is Windows 2000 or later, the RAS Connection Manager ignores the **dwDialMode, dwDialExtraPercent**, **dwDialExtraSampleSeconds**, **dwHangUpExtraPercent**, and **dwHangUpExtraSampleSeconds** members. RAS uses these members for the Bandwidth Allocation Protocol (BAP). BAP is available only on Windows 2000 or later versions.

Windows 2000 and later: If the RAS client is using Bandwidth Allocation Protocol (BAP) with server callback, the registry value **BapListenTimeout** specifies the length of time, in seconds, the client will wait for the server to callback. This value is located beneath the registry key:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RasMan\ppp

BapListenTimeout is of type **REG_DWORD**. **BapListenTimeout** can be any number in the range 0 to 0xFFFFFFFF. It has a default value of 30.

Windows 2000 and later: If **dwEncryptionType** is ET_None, but RASEO_RequireDataEncryption is specified, it is as though **dwEncryptionType** was ET_Require.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 OSR2 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RASADFunc**, **RasGetCountryInfo**, **RasGetEntryProperties**, **RasSetEntryProperties**, **RasSetSubEntryProperties**

RASENTRYDLG

The **RASENTRYDLG** structure is used in the **RasEntryDlg** function to specify additional input and output parameters.

```
typedef struct tagRASENTRYDLG {
  IN  DWORD dwSize;
  IN  HWND  hwndOwner;
  IN  DWORD dwFlags;
  IN  LONG  xDlg;
  IN  LONG  yDlg;
  OUT WCHAR szEntry[ RAS_MaxEntryName + 1 ];
  OUT DWORD dwError;
  IN  DWORD reserved;
  IN  DWORD reserved2;
} RASENTRYDLG;
```

Members

dwSize

Specifies the size of this structure, in bytes. Before calling **RasEntryDlg**, set this member to **sizeof(RASENTRYDLG)** to indicate the version of the structure. If **dwSize** is not a valid size, **RasEntryDlg** fails and sets the **dwError** member to ERROR_INVALID_SIZE.

hwndOwner

Specifies the window that owns the modal **RasEntryDlg** dialog box. This member can be any valid window handle, or it can be NULL if the dialog box has no owner.

dwFlags

A set of bit flags that indicate the options enabled for the dialog box. This parameter can be a combination of the RASEDFLAG_PositionDlg flag and one of the other flags listed following to indicate whether the **RasEntryDlg** function is creating, copying, or editing a phone-book entry.

Value	Meaning
RASEDFLAG_PositionDlg	Causes **RasEntryDlg** to use the values specified by the **xDlg** and **yDlg** members to position the dialog box. If this flag is not set, the dialog box is centered on the owner window, unless **hwndOwner** is NULL, in which case, the dialog box is centered on the screen.
RASEDFLAG_NewEntry	Causes **RasEntryDlg** to display a wizard for creating a new phone-book entry.
RASEDFLAG_CloneEntry	Causes **RasEntryDlg** to create a new entry by copying the properties of an existing entry. The function displays a property sheet containing the properties associated with the phone-book entry specified by the *lpszEntry* parameter of **RasEntryDlg**. The user can edit the properties and specify a name for the new entry.
RASEDFLAG_NoRename	Causes **RasEntryDlg** to display a property sheet for editing the properties of the phone-book entry specified by the *lpszEntry* parameter of **RasEntryDlg**. The user can change the properties of the entry but not its name.

xDlg
Specifies the horizontal screen coordinate of the upper-left corner of the dialog box. This value is used only if the RASEDFLAG_PositionDlg flag is set.

yDlg
Specifies the vertical screen coordinate of the upper-left corner of the dialog box. This value is used only if the RASEDFLAG_PositionDlg flag is set.

szEntry
On exit, **szEntry** is set to the name of the phone-book entry that was edited or created.

dwError
The **RasEntryDlg** function sets this member to a system error code or RAS error code if an error occurs. If no error occurs, the function sets **dwError** to zero. This value is ignored on input.

reserved
Reserved; must be zero.

reserved2
Reserved; must be zero.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasEntryDlg**

RASENTRYNAME

The **RASENTRYNAME** structure contains an entry name from a remote access phone book. The **RasEnumEntries** function returns an array of these structures.

```
typedef struct _RASENTRYNAME {
  DWORD  dwSize;
  TCHAR  szEntryName[RAS_MaxEntryName + 1];
#if (WINVER >= 0x500)
  DWORD dwFlags;
  CHAR  szPhonebookPath[MAX_PATH + 1];
#endif
} RASENTRYNAME;
```

Members

dwSize

Specifies the structure size, in bytes. Before using **RASENTRYNAME** in a function call, set this member to **sizeof(RASENTRYNAME)**.

szEntryName

Specifies a string containing the name of a remote access phone-book entry.

dwFlags

Windows 2000: Specifies whether the entry is in the system phone book in the AllUsers profile, or in the user's profile phone book. This member should be one of the following values.

Value	Meaning
REN_AllUsers	The phone book is a system phone book and is in the AllUsers profile.
REN_User	The phone book is in the user's profile.

szPhonebookPath

Windows 2000: Specifies the full path and file name of the Phone-Book (PBK) file.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasEnumEntries**

RASIPADDR

The **RASIPADDR** structure contains an IP address. The **RASENTRY** structure uses this structure to specify the IP addresses of various servers associated with an entry in a RAS phone book.

```
typedef struct RASIPADDR {
  BYTE    a;
  BYTE    b;
  BYTE    c;
  BYTE    d;
} RASIPADDR;
```

Members

a, **b**, **c**, and **d**

These members specify the value of the corresponding location in the "a.b.c.d" IP address.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RASENTRY**

RASMONITORDLG

The **RASMONITORDLG** structure is used in the **RasMonitorDlg** function to specify additional input and output parameters.

```
typedef struct tagRASMONITORDLG {
  IN  DWORD dwSize;
  IN  HWND  hwndOwner;
  IN  DWORD dwFlags;
  IN  DWORD dwStartPage;
  IN  LONG  xDlg;
  IN  LONG  yDlg;
  OUT DWORD dwError;
  IN  DWORD reserved;
  IN  DWORD reserved2;
} RASMONITORDLG;
```

Members

dwSize

Specifies the size of this structure, in bytes. Before calling **RasMonitorDlg**, set this member to **sizeof(RASMONITORDLG)** to indicate the version of the structure. If **dwSize** is not a valid size, **RasMonitorDlg** fails and sets the **dwError** member to ERROR_INVALID_SIZE.

hwndOwner

Specifies the window that owns the modal **RasMonitorDlg** property sheet. This member can be any valid window handle, or it can be NULL if the property sheet has no owner.

dwFlags

A bit flag that indicates the options that are enabled for the property sheet. You can specify the following value.

Value	Meaning
RASMDFLAG_PositionDlg	Causes **RasMonitorDlg** to use the values specified by the **xDlg** and **yDlg** members to position the dialog box. If this flag is not set, the dialog box is centered on the owner window, unless **hwndOwner** is NULL, in which case, the dialog box is centered on the screen.

dwStartPage

A set of bit flags that indicate the initial page of the property sheet to display on top. You can specify one of the following values.

Value	Meaning
RASMDPAGE_Status	Display the **Status** page on top. This is the default.
RASMDPAGE_Summary	Display the **Summary** page on top.
RASMDPAGE_Preferences	Display the **Preferences** page on top.

xDlg

Specifies the horizontal screen coordinate of the upper-left corner of the property sheet. This value is used only if the RASMDFLAG_PositionDlg flag is set.

yDlg

Specifies the vertical screen coordinate of the upper-left corner of the property sheet. This value is used only if the RASMDFLAG_PositionDlg flag is set.

dwError

The **RasMonitorDlg** function sets this member to a system error code or RAS error code if an error occurs. If no error occurs, the function sets **dwError** to zero. This value is ignored on input.

reserved

Reserved; must be zero.

reserved2

Reserved; must be zero.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasMonitorDlg**

RASNOUSER

The **RASNOUSER** structure is used with the **RasPBDlgFunc** callback function to specify authentication credentials and other information. This structure enables dial-up networking operations that begin before a user has logged on. It is provided to support the WinLogon application, and is not typically used by other applications.

```
typedef struct tagRASNOUSER {
  IN  DWORD dwSize;
  IN  DWORD dwFlags;
  OUT DWORD dwTimeoutMs;
  OUT TCHAR szUserName[ UNLEN + 1 ];
  OUT TCHAR szPassword[ PWLEN + 1 ];
  OUT TCHAR szDomain[ DNLEN + 1 ];
} RASNOUSER;
```

Members

dwSize

Specifies the size of this structure, in bytes. This member indicates the version of the structure.

dwFlags

Reserved; must be zero.

dwTimeoutMs

Specifies the time, in milliseconds, before the **RasPhonebookDlg** dialog box closes and returns to the caller as if the user had pressed the **Close** button. This feature is required for code that displays a window during WinLogon. If the user leaves his or her terminal for some time, the dialog box closes and WinLogon reverts to the CTRL+ALT+DEL prompt.

szUserName

Specifies a null-terminated string that contains the name of the user. This string is used to authenticate the user's right to access the remote access server.

szPassword

Specifies a null-terminated string that contains the user's password. This string is used to authenticate the user's right to access the remote access server.

szDomain

Specifies a null-terminated string that contains the domain on which authentication is to occur. An empty string ("") specifies the domain in which the remote access server is a member.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasPBDlgFunc**, **RasPhonebookDlg**

RASPBDLG

The **RASPBDLG** structure is used with the **RasPhonebookDlg** function to specify additional input and output parameters.

```
typedef struct tagRASPBDLG {
  IN  DWORD         dwSize;
  IN  HWND          hwndOwner;
  IN  DWORD         dwFlags;
  IN  LONG          xDlg;
  IN  LONG          yDlg;
  IN  DWORD         dwCallbackId;
  IN  RASPBDLGFUNC  pCallback;
  OUT DWORD         dwError;
  IN  DWORD         reserved;
  IN  DWORD         reserved2;
} RASPBDLG;
```

Members

dwSize

Specifies the size of this structure, in bytes. Before calling **RasPhonebookDlg**, set this member to **sizeof(RASPBDLG)** to indicate the version of the structure. If **dwSize** is not a valid size, **RasPhonebookDlg** fails and sets the **dwError** member to ERROR_INVALID_SIZE.

hwndOwner

Specifies the window that owns the modal **RasPhonebookDlg** dialog box. This member can be any valid window handle, or it can be NULL if the dialog box has no owner.

dwFlags

A set of bit flags that indicate the options enabled for the dialog box. This parameter can be a combination of the following values.

Value	Meaning
RASPBDFLAG_ PositionDlg	Causes **RasPhonebookDlg** to use the values specified by the **xDlg** and **yDlg** members to position the dialog box. If this flag is not set, the dialog box is centered on the owner window, unless **hwndOwner** is NULL, in which case, the dialog box is centered on the screen.
RASPBDFLAG_ ForceCloseOnDial	Turns on the close-on-dial option, overriding the user's preference. This option is appropriate with features such as RAS AutoDial where the user's goal is to make a connection immediately.

(continued)

(continued)

Value	Meaning
RASPBDFLAG_ NoUser	Causes the **RasPBDlgFunc** callback function specified by the **pCallback** member to receive a RASPBDEVENT_NoUser notification when the dialog box is starting up. This flag is for use in situations in which there is no logged-on user, as in the WinLogon application. Typically, applications should not use this flag.
RASPBDFLAG_ UpdateDefaults	Causes the default window position to be saved on exit. This flag is used primarily by RASPHONE.EXE and should not be used by typical applications.

xDlg

Specifies the horizontal screen coordinate of the upper-left corner of the dialog box. This value is used only if the RASPBDFLAG_PositionDlg flag is set.

yDlg

Specifies the vertical screen coordinate of the upper-left corner of the dialog box. This value is used only if the RASPBDFLAG_PositionDlg flag is set.

dwCallbackId

Specifies an application-defined value that is passed to the callback function specified by **pCallback**. You can use **dwCallbackId** to pass a pointer to application-specific context information.

pCallback

Pointer to a **RasPBDlgFunc** callback function that receives notifications of user activity while the dialog box is open. This member can be NULL if you do not want notifications.

dwError

The **RasPhonebookDlg** function sets this member to a system error code or RAS error code if an error occurs. If no error occurs, the function sets **dwError** to zero. This value is ignored on input.

reserved

Reserved; must be zero.

reserved2

Reserved; must be zero.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasdlg.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasPBDlgFunc**, **RasPhonebookDlg**

RASPPPCCP

The **RASPPPCCP** structure contains information that describes the results of a Compression Control Protocol (CCP) negotiation.

```
typedef struct tagRASPPPCCP {
  DWORD    dwSize;
  DWORD    dwError;
  DWORD    dwCompressionAlgorithm;
  DWORD    dwOptions;
  DWORD    dwServerCompressionAlgorithm;
  DWORD    dwServerOptions;
} RASPPPCCP;
```

Members

dwSize

Size of the **RASPPPCCP** structure. Ensure that this member contains the size of the structure before using the structure in a function call.

dwError

If the negotiation was unsuccessful, **dwError** contains the error that occurred.

dwCompressionAlgorithm

The compression algorithm in use by the client. The following table shows the possible values for this member.

Value	Meaning
RASCCPCA_MPPC	Microsoft Point to Point Compression (MPPC) Protocol (RFC 2118)
RASCCPCA_STAC	STAC option 4 (RFC 1974)

dwOptions

Specifies the compression options on the client. The following options are supported.

Option	Meaning
RASCCPO_Compression	Compression without encryption.
RASCCPO_HistoryLess	Microsoft Point to Point Encryption (MPPE) in stateless mode. The session key is changed after every packet. This mode improves performance on high latency networks, or networks that experience significant packet loss.

(continued)

(continued)

Option	Meaning
RASCCPO_Encryption56bit	MPPE using 56-bit keys.
RASCCPO_Encryption40bit	MPPE using 40-bit keys.
RASCCPO_Encryption128bit	MPPE using 128-bit keys.

The last three options are used when a connection is made over Layer 2 Tunneling Protocol (L2TP), and the connection uses IPSec encryption.

dwServerCompressionAlgorithm

The compression algorithm in use by the server. The following table shows the possible values for this member.

Value	Meaning
RASCCPCA_MPPC	Microsoft Point to Point Compression (MPPC) Protocol
RASCCPCA_STAC	STAC option 4

dwServerOptions

Specifies the compression options on the server. The following options are supported.

Option	Meaning
RASCCPO_Compression	Compression without encryption.
RASCCPO_HistoryLess	Microsoft Point to Point Encryption (MPPE) in stateless mode. The session key is changed after every packet. This mode improves performance on high latency networks, or networks that experience significant packet loss.
RASCCPO_Encryption56bit	MPPE using 56-bit keys.
RASCCPO_Encryption40bit	MPPE using 56-bit keys.
RASCCPO_Encryption128bit	MPPE using 56-bit keys.

The last three options are used when a connection is made over Layer 2 Tunneling Protocol (L2TP), and the connection uses IPSec encryption.

! Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetProjectionInfo**, **RASPROJECTION**, **RASPPPLCP**

RASPPPIP

The **RASPPPIP** structure contains the result of a PPP IP projection operation.

The **RasGetProjectionInfo** function returns a **RASPPPIP** data structure when its *rasprojection* parameter has the value RASP_PppIp.

```
typedef struct _RASPPPIP {
  DWORD    dwSize;
  DWORD    dwError;
  TCHAR    szIpAddress[ RAS_MaxIpAddress + 1 ];
#ifndef WINNT35COMPATIBLE
  TCHAR    szServerIpAddress[ RAS_MaxIpAddress + 1 ];
#endif
#if (WINVER >= 0x500)
  DWORD    dwOptions;
  DWORD    dwServerOptions;
#endif
} RASPPPIP;
```

Members

dwSize

Specifies the size of the structure, in bytes. Before calling the **RasGetProjectionInfo** function, set this member to indicate the version of the **RASPPPIP** structure that you are using. For information about earlier versions of this structure, see the following Remarks section.

dwError

Contains the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation, the error that prevented the projection from completing successfully.

szIpAddress

Contains a zero-terminated string that is the client's IP address on the RAS connection. This address string has the form *a.b.c.d*; for example, "11.101.237.71".

szServerIpAddress

Contains a null-terminated string that is the IP address of the remote PPP peer (that is, the server's IP address). This string is in "a.b.c.d" form. PPP does not require that servers provide this address, but Windows NT/Windows 2000 servers will consistently return the address anyway. Other PPP vendors may not provide the address. If the address is not available, this member returns an empty string, "".

dwOptions

Windows 2000 and later: Specifies IPCP options for the local computer. Currently, the only option is RASIPO_VJ. This option indicates that IP datagrams sent by the local computer are compressed using Van Jacobson compression.

dwServerOptions

Windows 2000 and later: Specifies IPCP options for the remote peer. Currently, the only option is RASIPO_VJ. This option indicates that IP datagrams sent by the remote peer (that is, received by the local computer) are compressed using Van Jacobson compression.

Remarks

The **szServerIpAddress** member was added to the **RASPPPIP** structure beginning with Windows NT 3.51 and the initial release of Windows 95. Beginning with these systems, **RasGetProjectionInfo** will support both the current form of the structure and the old form without the **szServerIpAddress** member. Use the **dwSize** member to indicate which version you are using.

For Windows NT 4.0 and earlier versions, **RasGetProjectionInfo** will return ERROR_INVALID_SIZE if **dwSize** specifies the current structure size. To retrieve PPP IP information from older systems, **dwSize** must specify the size of the old structure without the **szServerIpAddress** member.

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetProjectionInfo**, **RASPROJECTION**

RASPPPLCP

The **RASPPPLCP** structure contains information that describes the results of a PPP Link Control Protocol (LCP)/multi-link negotiation.

```
typedef struct tagRASPPPLCP {
  DWORD dwSize;
  BOOL  fBundled;
#if (WINVER >= 0x500)
  DWORD  dwError;
  DWORD  dwAuthenticationProtocol;
  DWORD  dwAuthenticationData;
  DWORD  dwEapTypeId;
  DWORD  dwServerAuthenticationProtocol;
```

```
  DWORD  dwServerAuthenticationData;
  DWORD  dwServerEapTypeId;
  BOOL   fMultilink;
  DWORD  dwTerminateReason;
  DWORD  dwServerTerminateReason;
  TCHAR  szReplyMessage[RAS_MaxReplyMessage];
  DWORD  dwOptions;
  DWORD  dwServerOptions;
#endif
} RASPPPLCP;
```

Members

dwSize

Size of the **RASPPPLCP** structure. Ensure that this member contains the size of the structure before using the structure in a function call.

fBundled

If this member is TRUE, the connection is composed of multiple links. Otherwise, this member is FALSE.

dwError

If the negotiation was unsuccessful, **dwError** contains the error that occurred.

dwAuthenticationProtocol

The authentication protocol used to authenticate the client. This member can be one of the following values.

Value	Meaning
RASLCPAP_PAP	Password Authentication Protocol
RASLCPAP_SPAP	Shiva Password Authentication Protocol
RASLCPAP_CHAP	Challenge Handshake Authentication Protocol
RASLCPAP_EAP	Extensible Authentication Protocol

dwAuthenticationData

Provides additional information about the authentication protocol specified by the **dwAuthenticationProtocol** member. This member can be one of the following values.

Value	Meaning
RASLCPAD_CHAP_MD5	MD5 CHAP
RASLCPAD_CHAP_MS	Microsoft CHAP
RASLCPAD_CHAP_MSV2	Microsoft CHAP version 2

dwEapTypeId

Provides the type ID of the extensible authentication protocol (EAP) used to authenticate the local computer. The value of this member is valid only if **dwAuthenticationProtocol** is RASLCPAPP_EAP.

dwServerAuthenticationProtocol
The authentication protocol used to authenticate the server. See the **dwAuthenticationProtocol** member for a list of possible values.

dwServerAuthenticationData
Provides additional information about the authentication protocol specified by **dwServerAuthenticationProtocol**. See the **dwAuthenticationData** member for a list of possible values.

dwServerEapTypeId
Provides the type ID of the extensible authentication protocol (EAP) used to authenticate the remote computer. The value of this member is valid only if **dwServerAuthenticationProtocol** is RASLCPAP_EAP.

fMultilink
If this member is TRUE, the connection supports multi-link. Otherwise, this member is FALSE.

dwTerminateReason
This member always has a value of zero.

dwServerTerminateReason
This member always has a value of zero.

szReplyMessage[RAS_MaxReplyMessage]
Pointer to a string that contains the message, if any, from the authentication protocol success/failure packet.

dwOptions
Provides additional LCP options for the local computer. This member is a combination of the following flags.

Flag	Meaning
RASLCPO_PFC	Protocol Field Compression (see *RFC 1172*)
RASLCPO_ACFC	Address and Control Field Compression (see *RFC 1172*)
RASLCPO_SSHF	Short Sequence Number Header Format (see *RFC 1990*)
RASLCPO_DES_56	DES 56-bit encryption
RASLCPO_3_DES	Triple DES Encryption

dwServerOptions
Provides addition LCP options for the remote computer. This member is a combination of the following flags.

Flag	Meaning
RASLCPO_PFC	Protocol Field Compression (see *RFC 1172*)
RASLCPO_ACFC	Address and Control Field Compression (see *RFC 1172*)
RASLCPO_SSHF	Short Sequence Number Header Format (see *RFC 1990*)
RASLCPO_DES_56	DES 56-bit encryption
RASLCPO_3_DES	Triple DES Encryption

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetProjectionInfo**, **RASPROJECTION**, **RASPPPCCP**

RASPPPIPX

The **RASPPPIPX** structure contains the result of a PPP IPX projection operation.

The **RasGetProjectionInfo** function returns a **RASPPPIPX** data structure when its *rasprojection* parameter has the value RASP_PppIpx.

```
typedef struct _RASPPPIPX {
  DWORD    dwSize;
  DWORD    dwError;
  TCHAR    szIpxAddress[ RAS_MaxIpxAddress + 1 ];
} RASPPPIPX;
```

Members

dwSize

Specifies the size of the structure, in bytes. Before calling the **RasGetProjectionInfo** function, set this member to **sizeof(RASPPPIPX)**. The function can then determine the version of the **RASPPPIPX** data structure that the caller of **RasGetProjectionInfo** is expecting. This allows backwards compatibility for compiled applications if there are future enhancements to the data structure.

dwError

Contains the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation, the error that prevented the projection from completing successfully.

szIpxAddress

Contains a zero-terminated string that is the client's IPX address on the RAS connection. This address string has the form *net.node*; for example, "1234ABCD.12AB34CD56EF".

Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetProjectionInfo**, **RASPROJECTION**

RASPPPNBF

The **RASPPPNBF** structure contains the result of a PPP NetBEUI Framer (NBF) projection operation.

The **RasGetProjectionInfo** function returns a **RASPPPNBF** data structure when its *rasprojection* parameter has the value RASP_PppNbf.

```
typedef struct _RASPPPNBF {
  DWORD   dwSize;
  DWORD   dwError;
  DWORD   dwNetBiosError;
  TCHAR   szNetBiosError[ NETBIOS_NAME_LEN + 1 ];
  TCHAR   szWorkstationName[ NETBIOS_NAME_LEN + 1 ];
  BYTE    bLana;
} RASPPPNBF;
```

Members

dwSize

Specifies the size of the structure, in bytes. Before calling the **RasGetProjectionInfo** function, set this member to **sizeof(RASPPPNBF)**. The function can then determine the version of the **RASPPPNBF** data structure that the caller of **RasGetProjectionInfo** is expecting. This allows backwards compatibility for compiled applications if there are future enhancements to the data structure.

dwError

Contains the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation, the error that prevented the projection from completing successfully.

dwNetBiosError

If **dwError** has the value ERROR_SERVER_NOT_RESPONDING or ERROR_NETBIOS_ERROR, the **dwNetBiosError** field contains the NetBIOS error that occurred. For other values of **dwError**, this field contains zero.

Windows 95: This member is undefined.

szNetBiosError

If **dwError** has the value ERROR_NAME_EXISTS_ON_NET, the **szNetBiosError** field contains a zero-terminated string that is the NetBIOS name that caused the conflict. For other values of **dwError**, this field contains the null string.

szWorkStationName

Contains a zero-terminated string that is the local workstation's computer name. This unique computer name is the closest NetBIOS equivalent to a client's NetBEUI address on a remote access connection.

bLana

Specifies the NetBIOS network adapter identifier, or LANA, on which the remote access connection was established. This member contains the value 0xFF if a connection was not established.

! Requirements

Windows NT/2000: Requires Windows NT 3.1 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

+ See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetProjectionInfo**, **RASPROJECTION**

RASSLIP

The **RASSLIP** structure contains the results of a the Serial Line Internet Protocol (SLIP) projection operation.

```
RASSLIP {
  DWORD dwSize;
  DWORD dwError;
  TCHAR szIpAddress[ RAS_MaxIpAddress + 1 ];
};
```

Members

dwSize

Specifies the size, in bytes, of the **RASSLIP** structure. Before calling the **RasGetProjectionInfo** function, set **dwSize** to `sizeof(RASSLIP)` to identify the version of the structure.

dwError

Specifies whether SLIP is configured. If **dwError** is zero, SLIP framing is configured. Otherwise, **dwError** is ERROR_PROTOCOL_NOT_CONFIGURED.

szIpAddress

A null-terminated string that contains the client's IP address on the RAS connection. This address string has the form *a.b.c.d*; for example, "11.101.237.71".

Remarks

If the **RASENTRY** structure for the phone-book entry used in a RAS connection specifies SLIP framing, you can call **RasGetProjectionInfo** with a **RASPROJECTION** of RASP_Slip to determine whether SLIP framing was successfully configured.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RASENTRY**, **RasGetProjectionInfo**, **RASPROJECTION**

RASSUBENTRY

The **RASSUBENTRY** structure contains information about a subentry of a RAS phone-book entry. The **RasSetSubEntryProperties** and **RasGetSubEntryProperties** functions use this structure to set and retrieve the properties of a subentry.

```
typedef struct tagRASSUBENTRY {
  DWORD       dwSize;
  DWORD       dwfFlags;
  //
  // Device
  //
  TCHAR       szDeviceType[ RAS_MaxDeviceType + 1 ];
```

```
    TCHAR      szDeviceName[ RAS_MaxDeviceName + 1 ];
    //
    // Phone numbers
    //
    TCHAR      szLocalPhoneNumber[ RAS_MaxPhoneNumber + 1 ];
    DWORD      dwAlternateOffset;

} RASSUBENTRY;
```

Members

dwSize

Specifies the size, in bytes, of the **RASSUBENTRY** structure. Before calling **RasSetSubEntryProperties** or **RasGetSubEntryProperties**, set **dwSize** to `sizeof(RASSUBENTRY)` to identify the version of the structure.

dwfFlags

Currently unused. The **RasSetSubEntryProperties** function sets this member to zero. The **RasGetSubEntryProperties** function ignores this member.

szDeviceType

Specifies a null-terminated string indicating the RAS device type referenced by **szDeviceName**. This member can be one of the following string constants.

String	Description
RASDT_Modem	A modem accessed through a COM port.
RASDT_Isdn	An ISDN card with the corresponding NDISWAN driver installed.
RASDT_X25	An X.25 card with the corresponding NDISWAN driver installed.
RASDT_Vpn	A virtual private network connection.
RASDT_Pad	A Packet Assembler/Disassembler

Windows 95: The RASDT_Vpn device type is supported on Windows 95 only if Microsoft Dial-Up Networking Version 1.2 is installed. The RASDT_X25 and RASDT_Pad device types are not supported on Windows 95.

Windows 98: The RASDT_Vpn device type is supported on Windows 98. However, the RASDT_X25 and RASDT_Pad device types are not currently supported on Windows 98

szDeviceName

Specifies a null-terminated string containing the name of the TAPI device to use with this phone-book entry. To enumerate all available RAS-capable devices, use the **RasEnumDevices** function.

szLocalPhoneNumber

Specifies a null-terminated string containing a telephone number. The way RAS uses this string depends on whether the RASEO_UseCountryAndAreaCodes flag is set in the **dwfOptions** member of the **RASENTRY** structure for this phone-book entry. If the flag is set, RAS combines **szLocalPhoneNumber** with the country and area codes specified in the **RASENTRY** structure. If the flag is not set, RAS uses the **szLocalPhoneNumber** string as the entire phone number.

dwAlternateOffset

Specifies the offset, in bytes, from the beginning of the structure to a list of consecutive null-terminated strings. The last string is terminated by two consecutive null characters. The strings are alternate phone numbers that RAS dials in the order listed if the primary number (see **szLocalPhoneNumber**) fails to connect. The alternate phone number strings are ANSI or Unicode, depending on whether you use the ANSI or Unicode version of the structure.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.
Unicode: Declared as Unicode and ANSI structures.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Structures, **RasGetSubEntryProperties**, **RasSetSubEntryProperties**

CHAPTER 9

RAS Message and Enumeration Types

Remote Access Service Message

Use WM_RASDIALEVENT to implement RAS functionality.

WM_RASDIALEVENT

The operating system sends a WM_RASDIALEVENT message to a window procedure when a change of state event occurs during a RAS connection process, and a window has been specified to handle notifications of such events by using the *notifier* parameter of **RasDial**.

The two message parameters are equivalent to the parameters of the same names that are used with **RasDialFunc** and **RasDialFunc1** callback functions.

```
WM_RASDIALEVENT
rasconnstate = (RASCONNSTATE) wParam;
      // connection state about to be entered
dwError = (DWORD) lParam;
      // error that may have occurred
```

Parameters

rasconnstate

Value of *wParam*. Equivalent to the *rasconnstate* parameter of the **RasDialFunc** and **RasDialFunc1** callback functions. Specifies a RASCONNSTATE enumerator value that indicates the state the RasDial remote access connection process is about to enter.

dwError

Value of *lParam*. Equivalent to the *dwError* parameter of the **RasDialFunc** and **RasDialFunc1** callback functions. A nonzero value indicates the error that has occurred, or zero if no error has occurred.

RasDial sends this message with *dwError* set to zero upon entry to each connection state. If an error occurs within a state, the message is sent again for the state, this time with a nonzero *dwError* value.

Return Values

If an application processes this message, it should return TRUE.

Requirements

Windows NT/2000: Requires Windows NT 3.5 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Messages, **RasDial**, **RasDialFunc**, **RasDialFunc1**, **RASCONNSTATE**

Remote Access Service Enumeration Types

RASCONNSTATE

The **RASCONNSTATE** enumeration type contains values that specify the states that may occur during a RAS connection operation. If you use the **RasDial** function to establish a RAS connection, you can specify a window, or a **RasDialFunc**, **RasDialFunc1**, or **RasDialFunc2** callback function to receive notification messages that report the current connection state. You can also use the **RasGetConnectStatus** function to get the connection state for a specified connection.

```
typedef enum _RASCONNSTATE {
  RASCS_OpenPort = 0,
  RASCS_PortOpened,
  RASCS_ConnectDevice,
  RASCS_DeviceConnected,
  RASCS_AllDevicesConnected,
  RASCS_Authenticate,
  RASCS_AuthNotify,
  RASCS_AuthRetry,
  RASCS_AuthCallback,
  RASCS_AuthChangePassword,
  RASCS_AuthProject,
  RASCS_AuthLinkSpeed,
  RASCS_AuthAck,
  RASCS_ReAuthenticate,
  RASCS_Authenticated,
  RASCS_PrepareForCallback,
  RASCS_WaitForModemReset,
```

```
    RASCS_WaitForCallback,
    RASCS_Projected,

#if (WINVER >= 0x400)
    RASCS_StartAuthentication,    // Windows 95 only
    RASCS_CallbackComplete,       // Windows 95 only
    RASCS_LogonNetwork,           // Windows 95 only
#endif
    RASCS_SubEntryConnected,
    RASCS_SubEntryDisconnected,

    RASCS_Interactive = RASCS_PAUSED,
    RASCS_RetryAuthentication,
    RASCS_CallbackSetByCaller,
    RASCS_PasswordExpired,
#if (WINVER >= 0x500)
    RASCS_InvokeEapUI,
#endif

    RASCS_Connected = RASCS_DONE,
    RASCS_Disconnected
} RASCONNSTATE ;
```

The enumerator values are listed here in the general order in which the connection states occur. However, you should not write code that depends on the order or occurrence of particular **RASCONNSTATE** connection states, because this may vary between platforms.

Enumerator Value	Meaning
RASCS_OpenPort	The communication port is about to be opened.
RASCS_PortOpened	The communication port has been opened successfully.
RASCS_ConnectDevice	A device is about to be connected. **RasGetConnectStatus** can be called to determine the name and type of the device being connected.
RASCS_DeviceConnected	A device has connected successfully. **RasGetConnectStatus** can be called to determine the name and type of the device being connected.

(continued)

(continued)

Enumerator Value	Meaning
	For a simple modem connection, RASCS_ConnectDevice and RASCS_DeviceConnected will be called only once. For a dial-up X.25 PAD connection, the pair will be called first for the modem, then for the PAD. If a preconnect switch is configured, the pair will be called for the switch before any other devices connect. Likewise, the pair will be called for a postconnect switch after any other devices connect. **Windows 95:** Note that Windows 95 does not currently support multistage connections such as the X.25 PAD connection described earlier.
RASCS_AllDevicesConnected	All devices in the device chain have successfully connected. At this point, the physical link is established.
RASCS_Authenticate	The authentication process is starting. Remote access does not allow the remote client to generate any traffic on the LAN until authentication has been successfully completed. Remote access authentication on a Windows NT/ Windows 2000 or Windows 95 server consists of: • Validating the user name/password on the specified domain. • Projecting the client onto the LAN. This means that the remote access server does what is necessary to send and receive data on the LAN on behalf of the client. For example, the remote access server might need to add a NetBIOS name that corresponds to the client's computer name. • Call-back processing in which the client hangs up and the server calls back. (The user needs special permissions on the remote access server for this.) • Calculating the link speed. This is necessary to correctly set transport time-outs to match the relatively slow speed of the remote link.
RASCS_AuthNotify	An authentication event has occurred. If *dwError* is zero, this event will be immediately followed by one of the more specific authentication states following. If *dwError* is nonzero, authentication has failed, and the error value indicates why.
RASCS_AuthRetry	The client has requested another validation attempt with a new user name/password/domain. This state does not occur in Windows NT version 3.1.
RASCS_AuthCallback	The remote access server has requested a callback number. This occurs only if the user has "Set By Caller" callback privilege on the server.

Enumerator Value	Meaning
RASCS_AuthChangePassword	The client has requested to change the password on the account. This state does not occur in Windows NT version 3.1.
RASCS_AuthProject	The projection phase is starting.
RASCS_AuthLinkSpeed	The link-speed calculation phase is starting.
RASCS_AuthAck	An authentication request is being acknowledged.
RASCS_ReAuthenticate	Reauthentication (after callback) is starting.
RASCS_Authenticated	The client has successfully completed authentication.
RASCS_PrepareForCallback	The line is about to disconnect in preparation for callback.
RASCS_WaitForModemReset	The client is delaying in order to give the modem time to reset itself in preparation for callback.
RASCS_WaitForCallback	The client is waiting for an incoming call from the remote access server.
RASCS_Projected	This state occurs after the RASCS_AuthProject state. It indicates that projection result information is available. You can access the projection result information by calling **RasGetProjectionInfo**.
RASCS_StartAuthentication	**Windows 95 only:** Indicates that user authentication is being initiated or retried.
RASCS_CallbackComplete	**Windows 95 only:** Indicates that the client has been called back and is about to resume authentication.
RASCS_LogonNetwork	**Windows 95 only:** Indicates that the client is logging on to the network.
RASCS_SubEntryConnected	When dialing a multilink phone-book entry, this state indicates that a subentry has been connected during the dialing process. The *dwSubEntry* parameter of a **RasDialFunc2** callback function indicates the index of the subentry. When the final state of all subentries in the phone-book entry has been determined, the connection state is RASCS_Connected if one or more subentries have been connected successfully.
RASCS_SubEntryDisconnected	When dialing a multilink phone-book entry, this state indicates that a subentry has been disconnected during the dialing process. The *dwSubEntry* parameter of a **RasDialFunc2** callback function indicates the index of the subentry.
RASCS_Interactive	This state corresponds to the terminal state supported by RASPHONE.EXE. This state does not occur in Windows NT version 3.1.
RASCS_RetryAuthentication	This state corresponds to the retry authentication state supported by RASPHONE.EXE. This state does not occur in Windows NT version 3.1.

(continued)

(continued)

Enumerator Value	Meaning
RASCS_CallbackSetByCaller	This state corresponds to the callback state supported by RASPHONE.EXE. This state does not occur in Windows NT version 3.1.
RASCS_PasswordExpired	This state corresponds to the change password state supported by RASPHONE.EXE. This state does not occur in Windows NT version 3.1.
RASCS_InvokeEapUI	An application can use this paused state to bring up a custom authentication UI. The application should call the **RasInvokeEapUI** function to invoke the custom UI. RASCS_InvokeEapUI is a paused state.
RASCS_Connected	Successful connection.
RASCS_Disconnected	Disconnection or failed connection.

Remarks

The connection process states are divided into three classes: running states, paused states, and terminal states.

An application can easily determine the class of a specific state by performing Boolean bit operations with the RASCS_PAUSED and RASCS_DONE bitmasks. Here are some examples:

```
fDoneState = (state & RASCS_DONE);
fPausedState = (state & RASCS_PAUSED);
fRunState = !(fDoneState || fPausedState);
```

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Enumeration Types, **RasDial**, **RasInvokeEapUI**, **RasGetConnectStatus**, **RasGetProjectionInfo**, **RASCONNSTATUS**

RASPROJECTION

The **RASPROJECTION** enumeration type defines values that specify a particular authentication protocol or Point-to-Point Protocol (PPP) control protocol. An application passes a value of this type to the **RasGetProjectionInfo** function to specify the protocol of interest.

```
typedef enum _RASPROJECTION {
  RASP_Amb = 0x10000,
  RASP_PppNbf = 0x803F,
  RASP_PppIpx = 0x802B,
  RASP_PppIp = 0x8021
  RASP_PppCcp = 0x80FD;
  RASP_PppLcp = 0xC021;
  RASP_Slip = 0x20000
} RASPROJECTION ;
```

Each of the **RASPROJECTION** enumerators has a corresponding data structure; the **RasGetProjectionInfo** function returns the specified information in a structure of that type.

Enumerator Value	Meaning
RASP_Amb	Specifies the Authentication Message Block (AMB) authentication protocol. AMB is a NetBIOS-based protocol used to authenticate with downlevel remote access servers (all those prior to Windows NT 3.5). The corresponding data structure is a **RASAMB**.
RASP_PppNbf	Specifies the NetBEUI Framer (NBF) protocol. NBFCP is a PPP network control protocol used to negotiate the parameters necessary to ship NetBEUI packets on a WAN link. The corresponding data structure is a **RASPPPNBF**.
RASP_PppIpx	Specifies the Internetwork Packet Exchange (IPX) control protocol. IPXCP is a PPP network control protocol used to negotiate the parameters necessary to ship IPX packets on a WAN link. The corresponding data structure is a **RASPPPIPX**.
RASP_PppIp	Specifies the Internet Protocol (IP) control protocol. IPCP is a PPP network control protocol used to negotiate the parameters necessary to ship IP packets on a WAN link. The corresponding data structure is a **RASPPPIP**.
RASP_PppCcp	Specifies the Compression Control Protocol (CCP). CCP enables computers using PPP to negotiate compression algorithms and parameters. The corresponding data structure is **RASPPPCCP**.
RASP_PppLcp	Specifies the Link Control Protocol (LCP). LCP is used by computers to establish, modify, and terminate PPP connections. The corresponding data structure is **RASPPPLCP**.
RASP_Slip	Specifies the Serial Line Internet Protocol (SLIP). SLIP is a framing protocol used primarily in UNIX environments.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Requires Windows 95 or later.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, Remote Access Service Enumeration Types, **RasGetProjectionInfo**, **RASAMB**, **RASPPPIP**, **RASPPPIPX**, **RASPPPNBF**

CHAPTER 10

RAS Server Administration Reference

RAS Server Administration Functions

For Microsoft® Windows NT® 4.0, use the following functions to implement RAS Server Administration functionality. Microsoft® Windows® 95 does not provide RAS server support.

RasAdminFreeBuffer

The **RasAdminFreeBuffer** function frees memory that was allocated by RAS on behalf of the caller.

```
DWORD RasAdminFreeBuffer(
  PVOID Pointer  // pointer to the buffer to free
);
```

Parameters

Pointer
Pointer to the buffer to be freed.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be the following error code.

Value	Meaning
ERROR_INVALID_PARAMETER	The *Pointer* parameter is invalid.

There is no extended error information for this function; do not call **GetLastError**.

Remarks

Use the **RasAdminFreeBuffer** function to free the buffers allocated by the **RasAdminPortEnum** and **RasAdminPortGetInfo** functions.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RasAdminPortEnum**, **RasAdminPortGetInfo**

RasAdminGetErrorString

The **RasAdminGetErrorString** function retrieves a message string that corresponds to a RAS error code returned by one of the RAS server administration (RasAdmin) functions. These message strings are retrieved from the RASMSG.DLL that is installed as part of RAS.

```
DWORD RasAdminGetErrorString (
  UINT ResourceId,      // error code to get message for
  WCHAR *lpszString,    // pointer to a buffer that receives
                        // the error string
  DWORD InBufSize       // size, in characters, of the buffer
);
```

Parameters

ResourceId

Specifies an error code returned by one of the RasAdmin functions. This value must be in the range of error codes from RASBASE to RASBASEEND that are defined in Raserror.h.

lpszString

Pointer to a buffer that receives the error message corresponding to the specified error code.

InBufSize

Specifies the size, in characters, of the *lpszString* buffer. Error messages are typically 80 characters or less; a buffer size of 512 characters is always adequate.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is an error code. This value can be a last error value set by the **LoadLibrary**, **GlobalAlloc**, or **LoadString** functions; or it can be one of the following error codes.

Value	Meaning
ERROR_INVALID_PARAMETER	The *ResourceId* or *lpszString* parameters are invalid.
ERROR_INSUFFICIENT_BUFFER	The size specified by the *InBufSize* parameter is too small.

There is no extended error information for this function; do not call **GetLastError**.

Remarks

The RasAdmin functions can return error codes that are not in the range supported by the **RasAdminGetErrorString** function. For example, the RasAdmin functions can return error codes that are defined in Lmerr.h and Winerror.h. Before calling **RasAdminGetErrorString**, verify that the error code is in the range RASBASE to RASBASEEND, as defined in Raserror.h.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **LoadLibrary**, **GlobalAlloc**, **LoadString**

RasAdminGetUserAccountServer

The **RasAdminGetUserAccountServer** function retrieves the name of the server that has the user account database. You can use the returned server name in the **RasAdminUserGetInfo** and **RasAdminUserSetInfo** functions to get or set information about a specified user.

```
DWORD RasAdminGetUserAccountServer(
  const WCHAR *lpszDomain,      // pointer to the name of
                                // the Windows NT/2000 domain
  const WCHAR *lpszServer,      // pointer to the name of
                                // the RAS server
  LPWSTR lpszUserAccountServer  // receives the name of
                                // the user account server
);
```

Parameters

lpszDomain

Pointer to a null-terminated Unicode string that contains the name of the domain to which the RAS server belongs. This parameter can be NULL if you are running your RAS administration application on a Windows NT/2000 Workstation or Server that is not participating in a Windows NT/2000 domain. If this parameter is NULL, the *lpszServer* parameter must be non-NULL.

lpszServer

Pointer to a null-terminated Unicode string that contains the name of the Windows NT/Windows 2000 RAS server. Specify the name with leading "\\" characters, in the form: *servername*. This parameter can be NULL if the *lpszDomain* parameter is not NULL.

lpszUserAccountServer

Pointer to a buffer that receives a null-terminated Unicode string containing the name of a domain controller that has the user account database. The buffer should be big enough to hold the server name (UNCLEN +1). The function prefixes the returned server name with leading "\\" characters, in the form: *servername*.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be the following error code.

Value	Meaning
ERROR_INVALID_PARAMETER	Both *lpszDomain* and *lpszServer* are NULL.

There is no extended error information for this function; do not call **GetLastError**.

Remarks

The **RasAdminGetUserAccountServer** function can obtain the name of the server with the user accounts database given the name of the RAS server, or the name of the domain in which the RAS server resides.

The *lpszDomain* parameter should specify a valid Windows NT/Windows 2000 domain name. If you are running your RAS administration application on a Windows NT/Windows 2000 Server that is not participating in a Windows NT/Windows 2000 domain (for example, the server is in its own work group), then set *lpszDomain* to NULL. In this case, the *lpszServer* parameter must specify the server name. To get the server name, call the **GetComputerName** function. Be sure to prefix the server name with the "\\" characters.

If the server name specified by *lpszServer* is a stand-alone Windows NT/Windows 2000 Server (that is, the server or workstation does not participate in a Windows NT/Windows 2000 domain), then the server name itself is returned in the *lpszUserAccountServer* buffer.

You can then use the name of the user account server in a call to the **NetQueryDisplayInformation** function to enumerate the users in the user account database.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **GetComputerName**, **RasAdminUserGetInfo**, **RasAdminUserSetInfo**

RasAdminPortClearStatistics

The **RasAdminPortClearStatistics** function resets the counters representing the various statistics reported by the **RasAdminPortGetInfo** function in the **RAS_PORT_STATISTICS** structure. The counters are reset to zero and start accumulating from then on.

```
DWORD RasAdminPortClearStatistics(
  const WCHAR *lpszServer,  // pointer to the server name
  const WCHAR *lpszPort     // pointer to the name of the
                            // port on the server
);
```

Parameters

lpszServer
Pointer to a null-terminated Unicode string that contains the name of the Windows NT/Windows 2000 RAS server. Specify the name with leading "\\" characters, in the form: *servername*.

lpszPort
Pointer to a null-terminated Unicode string that contains the name of the port on the server.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be the following error code.

Value	Meaning
ERROR_DEV_NOT_EXIST	The specified port is invalid.

There is no extended error information for this function; do not call **GetLastError**.

Remarks

The **RasAdminPortClearStatistics** function clears the statistics on the server, not locally within the application that makes the call. This means that the statistics are also reset for any other application that is monitoring the specified port.

If the *lpszPort* port is part of a multilink connection, **RasAdminPortClearStatistics** resets the statistics for the specified port, The function also resets the cumulative statistics for the multilink connection. However, the function does not effect the individual statistics for other ports that are part of the multilink connection.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RAS_PORT_STATISTICS**, **RasAdminPortGetInfo**

RasAdminPortDisconnect

The **RasAdminPortDisconnect** function disconnects a port that is currently in use.

```
DWORD RasAdminPortDisconnect(
  const WCHAR *lpszServer,  // pointer to the server name
  const WCHAR *lpszPort     // pointer to the name of the
                            // port on the server
);
```

Parameters

lpszServer
Pointer to a null-terminated Unicode string that contains the name of the Windows NT/Windows 2000 RAS server. Specify the name with leading "\\" characters, in the form: *servername*.

lpszPort
Pointer to a null-terminated Unicode string that contains the name of the port on the server.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_INVALID_PORT	The specified port is invalid.
NERR_UserNotFound	The port is not currently in use.

There is no extended error information for this function; do not call **GetLastError**.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions

RasAdminPortEnum

The **RasAdminPortEnum** function enumerates all ports on the specified RAS server. For each port on the server, the function returns a **RAS_PORT_0** structure that contains information about the port.

```
DWORD RasAdminPortEnum(
  const WCHAR *lpszServer,   // pointer to the server name
  PRAS_PORT_0 *ppRasPort0,   // receives a pointer to an
                             // array of port information
  WORD *pcEntriesRead        // receives the number of ports
                             // enumerated
);
```

Parameters

lpszServer
Pointer to a null-terminated Unicode string that contains the name of the Windows NT/Windows 2000 RAS server. Specify the name with leading "\\" characters, in the form: *servername*.

ppRasPort0
Pointer to a variable that receives a pointer to a buffer that contains an array of **RAS_PORT_0** structures. When your application has finished with the memory, free it by calling the **RasAdminFreeBuffer** function.

pcEntriesRead
Pointer to a 16-bit variable that receives the total number of **RAS_PORT_0** structures returned in the *ppRasPort0* array.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be the following error code.

Value	Meaning
NERR_ItemNotFound	No ports could be enumerated. This could be because all configured ports on the server are currently being used for dialing out.

There is no extended error information for this function; do not call **GetLastError**.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RAS_PORT_0**, **RasAdminFreeBuffer**

RasAdminPortGetInfo

The **RasAdminPortGetInfo** function retrieves information about a specified port on a specified server.

```
DWORD RasAdminPortGetInfo(
  const WCHAR *lpszServer, // pointer to the server name
  const WCHAR *lpszPort,   // pointer to the name of port
                           // on the server
  RAS_PORT_1 *pRasPort1,   // receives the state of the port
  RAS_PORT_STATISTICS *pRasStats,
                           // receives statistics about
                           // the port
  RAS_PARAMETERS **ppRasParams
                           // receives an array of media-
                           // specific parameters and values
);
```

Parameters

lpszServer

Pointer to a null-terminated Unicode string that contains the name of the Windows NT/Windows 2000 RAS server. Specify the name with leading "\\" characters, in the form: *servername*.

lpszPort

Pointer to a null-terminated Unicode string that contains the name of the port on the server.

pRasPort1

Pointer to a **RAS_PORT_1** structure that the function fills in with information about the state of the port.

pRasStats

Pointer to a **RAS_PORT_STATISTICS** structure that the function fills in with statistics about the port.

ppRasParams

Pointer to a variable that receives a pointer to an array of **RAS_PARAMETERS** structures. Each structure contains the name of a media-specific key, such as MAXCONNECTBPS, and its associated value. When your application is finished with the memory pointed to by **ppRasParams*, free it by calling the **RasAdminFreeBuffer** function.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be one of the following error codes.

Value	Meaning
ERROR_DEV_NOT_EXIST	The specified port is invalid.
ERROR_NOT_ENOUGH_MEMORY	Insufficient memory to allocate a buffer for the *ppRasParams* array.

There is no extended error information for this function; do not call **GetLastError**.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RAS_PARAMETERS**, **RAS_PORT_1**, **RAS_PORT_STATISTICS**, **RasAdminFreeBuffer**

RasAdminServerGetInfo

The **RasAdminServerGetInfo** function gets the server configuration of a RAS server.

```
DWORD RasAdminServerGetInfo(
  const WCHAR *lpszServer,
          // pointer to the name of the RAS server
  PRAS_SERVER_0 pRasServer0
          // pointer to server information
);
```

Parameters

lpszServer

Pointer to a null-terminated Unicode string that contains the name of the Windows NT/Windows 2000 RAS server. If this parameter is NULL, the function returns information about the local computer. Specify the name with leading "\\" characters, in the form: *servername*.

pRasServer0

Pointer to a **RAS_SERVER_0** structure that receives the number of ports configured on the server, the number of ports currently in use, and the server version number.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value is an error code. Possible error codes include those returned by **GetLastError** for the **CallNamedPipe** function. There is no extended error information for this function; do not call **GetLastError**.

Remarks

To enumerate all RAS servers in a Windows NT/Windows 2000 domain, call the **NetServerEnum** function and specify SV_TYPE_DIALIN for the *servertype* parameter.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **NetServerEnum**, **RAS_SERVER_0**

RasAdminUserGetInfo

The **RasAdminUserGetInfo** function gets the RAS permissions and callback phone number information for a specified user.

```
DWORD RasAdminUserGetInfo(
  const WCHAR *lpszUserAccountServer,
                       // pointer to the name of the user
                       // account server
  const WCHAR *lpszUser, // pointer to the name of the user
  PRAS_USER_0 pRasUser0  // receives the user's RAS
                       // information
);
```

Parameters

lpszUserAccountServer
Pointer to a null-terminated Unicode string that contains the name of the primary or backup domain controller that has the user account database. Use the **RasAdminGetUserAccountServer** function to get this server name.

lpszUser
Pointer to a null-terminated Unicode string that contains the name of the user for whom to get RAS information.

pRasUser0
Pointer to a **RAS_USER_0** structure that receives the RAS data for the specified user.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be the following error code.

Value	Meaning
NERR_BufTooSmall	Insufficient memory to perform this function.

There is no extended error information for this function; do not call **GetLastError**.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RAS_USER_0**, **RasAdminGetUserAccountServer**, **RasAdminUserSetInfo**

RasAdminUserSetInfo

The **RasAdminUserSetInfo** function sets the RAS permissions and call-back phone number for a specified user.

```
DWORD RasAdminUserSetInfo(
  const WCHAR *lpszUserAccountServer,
                              // pointer to the name of
                              // the user account server
  const WCHAR *lpszUser,      // pointer to the name of
                              // the user
  const PRAS_USER_0 pRasUser0 // pointer to the new RAS
                              // information for this user
);
```

Parameters

lpszUserAccountServer
Pointer to a null-terminated Unicode string that contains the name of the primary or backup domain controller that has the user account database. Use the **RasAdminGetUserAccountServer** function to get this server name.

lpszUser
Pointer to a null-terminated Unicode string that contains the name of the user for whom RAS information is to be set.

pRasUser0
Pointer to a **RAS_USER_0** structure that contains the new RAS data for the specified user.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS.

If the function fails, the return value can be one of the following error codes.

Value	Description
ERROR_INVALID_DATA	The *pRasUser0* buffer contains invalid data.
ERROR_INVALID_CALLBACK_NUMBER	The callback number specified in the *pRasUser0* buffer contains invalid characters.
NERR_BufTooSmall	Insufficient memory to perform this function.

There is no extended error information for this function; do not call **GetLastError**.

Remarks

When setting the RAS permissions for a user, the **bfPrivilege** member of the **RAS_USER_0** structure must specify at least one of the call-back flags. For example, to set a user's privileges to allow dial-in privilege but no call-back privilege, set **bfPrivilege** to RASPRIV_DialinPrivilege | RASPRIV_NoCallback.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.
Library: Use Rassapi.lib.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RAS_USER_0**, **RasAdminGetUserAccountServer**, **RasAdminUserGetInfo**

RAS Administration DLL Functions

Implement the following functions when developing a RAS administration DLL:

RasAdminAcceptNewConnection
RasAdminConnectionHangupNotification
RasAdminGetIpAddressForUser
RasAdminReleaseIpAddress

RasAdminAcceptNewConnection

The **RasAdminAcceptNewConnection** function is an application-defined function that is exported by a third-party RAS server administration DLL. RAS calls this function when a user tries to establish a remote connection to a RAS server. The function decides whether the user is allowed to connect.

The RAS server calls **RasAdminAcceptNewConnection** once for each port in a multilink connection.

```
BOOL RasAdminAcceptNewConnection(
  RAS_PORT_1 *pRasPort1,       // pointer to information about
                               // the connection
  RAS_PORT_STATISTICS *pRasStats,
                               // pointer to statistics about
                               // the port
  RAS_PARAMETERS *pRasParams   // pointer to an array of
                               // media-specific parameters
                               // and values
);
```

Parameters

pRasPort1
: Pointer to a **RAS_PORT_1** structure that contains RAS data about the pending connection. This structure contains the relevant connection information that you need to make a decision about the connection.

pRasStats
Pointer to a **RAS_PORT_STATISTICS** structure that contains statistics about the port.

pRasParams
Pointer to an array of **RAS_PARAMETERS** structures. Each structure contains the name of a media-specific key, such as MAXCONNECTBPS, and its associated value.

Return Values

If the function returns TRUE, RAS accepts the new connection.

If the function returns FALSE, RAS does not accept the new connection. There is no extended error information for this function; do not call **GetLastError**.

Remarks

The **RasAdminAcceptNewConnection** function gives more control to a RAS server administration DLL to determine whether a specified remote user should be allowed to connect to a server.

An additional application of **RasAdminAcceptNewConnection** would be to send a popup message to newly connected clients. Use the **NetMessageBufferSend** function to send the message to the client computer.

The setup program for a third-party RAS administration DLL must register the DLL with RAS by providing information under the following key in the registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll

To register the DLL, set the following values under this key.

Value name	Value data
DisplayName	A REG_SZ string that contains the user-friendly display name of the DLL.
DLLPath	A REG_SZ string that contains the full path of the DLL.

For example, the registry entry for a RAS administration DLL from a fictional company named Netwerks Corporation might be:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll
DisplayName : REG_SZ : Netwerks RAS Admin DLL
DLLPath : REG_SZ : C:\nt\system32\ntwkadm.dll

The setup program for a RAS administration DLL should also provide remove/uninstall functionality. If a user removes the DLL, the setup program should delete the DLL's registry entries.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RAS_PARAMETERS**, **RAS_PORT_1**, **RAS_PORT_STATISTICS**

RasAdminConnectionHangupNotification

The **RasAdminConnectionHangupNotification** function is an application-defined function that is exported by a third-party RAS server administration DLL. When RAS disconnects an existing connection, it calls this function to notify your DLL.

The RAS server calls **RasAdminConnectionHangupNotification** once for each port in a multilink connection.

```
VOID RasAdminConnectionHangupNotification(
  RAS_PORT_1 *pRasPort1,  // pointer to information about
                          // the connection
  RAS_PORT_STATISTICS *pRasStats,
                          // pointer to statistics about
                           // the port
  RAS_PARAMETERS *pRasParams  // pointer to an array of
                              // media-specific parameters
                              // and values
);
```

Parameters

pRasPort1
Pointer to a **RAS_PORT_1** structure that contains RAS data about the connection that ended. This structure contains the relevant connection information that you can use to determine how long the port was connected.

pRasStats
Pointer to a **RAS_PORT_STATISTICS** structure that contains statistics about the port. RAS began accumulating these statistics when the connection was first established.

pRasParams
Pointer to an array of **RAS_PARAMETERS** structures. Each structure contains the name of a media-specific key, such as MAXCONNECTBPS, and its associated value.

Return Values

The function does not return a value. There are no extended error information for this function; do not call **GetLastError**.

Remarks

The RAS call to the **RasAdminConnectionHangupNotification** function is just a notification; no action is required from your DLL. You can use the information provided by this function for accounting purposes.

The setup program for a third-party RAS administration DLL must register the DLL with RAS by providing information under the following key in the registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll

To register the DLL, set the following values under this key.

Value name	Value data
DisplayName	A REG_SZ string that contains the user-friendly display name of the DLL.
DLLPath	A REG_SZ string that contains the full path of the DLL.

For example, the registry entry for a RAS administration DLL from a fictional company named Netwerks Corporation might be:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll
DisplayName : REG_SZ : Netwerks RAS Admin DLL
DLLPath : REG_SZ : C:\nt\system32\ntwkadm.dll

The setup program for a RAS administration DLL should also provide remove/uninstall functionality. If a user removes the DLL, the setup program should delete the DLL's registry entries.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RAS_PARAMETERS**, **RAS_PORT_1**, **RAS_PORT_STATISTICS**

RasAdminGetIpAddressForUser

The **RasAdminGetIpAddressForUser** function is an application-defined function that is exported by a third-party RAS server administration DLL. RAS calls this function to get an IP address for the dialed-in remote client.

```
DWORD RasAdminGetIpAddressForUser(
  WCHAR *lpszUserName, // pointer to the name of the
                       // remote user
  WCHAR *lpszPortName, // pointer to the name of the port
  IPADDR *pipAddress,  // pointer to the IP address
  BOOL *bNotifyRelease // indicates whether RAS should call
                       // RasAdminReleaseIpAddress
);
```

Parameters

lpszUserName
Pointer to a null-terminated Unicode string that contains the name of the remote user for whom an IP address is required.

lpszPortName
Pointer to a null-terminated Unicode string that contains the name of the port on which the user specified by *lpszUserName* is attempting to connect.

pipAddress
Pointer to an **IPADDR** variable. On input, **pipAddress* contains either zero or the IP address that the RAS server proposes to use for the dialed-in remote client. The function can set **pipAddress* to a different IP address, or accept the passed-in IP address. If **pipAddress* is zero on input, the function must provide an IP address; otherwise, the client will be unable to connect to this server using IP.

bNotifyRelease
Pointer to a **BOOL** variable. Set this variable to TRUE if you want RAS to call your **RasAdminReleaseIpAddress** function when the user disconnects from this port; otherwise, set it to FALSE.

Return Values

If *pipAddress* points to an IP address that the client can use to connect to this RAS server, the function should return NO_ERROR. This can occur if the function accepts the IP address that was passed by the RAS server, or if the function provides a different IP address.

If *pipAddress* does not point to an IP address, the function should return a nonzero error code. This can occur if no IP address is available, or if the passed in IP address is unacceptable. In this case, the client will be unable to connect to this server using IP. There is no extended error information for this function; do not call **GetLastError**.

Remarks

The setup program for a third-party RAS administration DLL must register the DLL with RAS by providing information under the following key in the registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll

To register the DLL, set the following values under this key.

Value name	Value data
DisplayName	A REG_SZ string that contains the user-friendly display name of the DLL.
DLLPath	A REG_SZ string that contains the full path of the DLL.

For example, the registry entry for a RAS administration DLL from a fictional company named Netwerks Corporation might be:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll
DisplayName : REG_SZ : Netwerks RAS Admin DLL
DLLPath : REG_SZ : C:\nt\system32\ntwkadm.dll

The setup program for a RAS administration DLL should also provide remove/uninstall functionality. If a user removes the DLL, the setup program should delete the DLL's registry entries.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RasAdminReleaseIpAddress**

RasAdminReleaseIpAddress

The **RasAdminReleaseIpAddress** function is an application-defined function that is exported by a third-party RAS server administration DLL. RAS calls this function to notify your DLL that the remote client was disconnected and that the IP address should be released.

```
VOID RasAdminReleaseIpAddress(
  WCHAR *lpszUserName,  // pointer to the name of the
                        // remote user
  WCHAR *lpszPortName,  // pointer to the name of the port
  IPADDR *pipAddress    // pointer to the IP address
);
```

Parameters

lpszUserName
Pointer to a null-terminated Unicode string that contains the name of a remote user for whom an IP address was previously obtained using the **RasAdminGetIpAddressForUser** function.

lpszPortName
Pointer to a null-terminated Unicode string that contains the name of the port on which the user specified by *lpszUserName* is connected.

pipAddress
Pointer to an **IPADDR** variable that contains the IP address returned for this user in a previous call to **RasAdminGetIpAddressForUser**.

Return Values

There is no extended error information for this function; do no call **GetLastError**.

Remarks

The RAS server calls your **RasAdminReleaseIpAddress** function only if your application returned TRUE in the *bNotifyRelease* parameter during the earlier call to **RasAdminGetIpAddressForUser** for the user specified by the *lpszUserName* parameter.

The setup program for a third-party RAS administration DLL must register the DLL with RAS by providing information under the following key in the registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll

To register the DLL, set the following values under this key.

Value name	Value data
DisplayName	A REG_SZ string that contains the user-friendly display name of the DLL.
DLLPath	A REG_SZ string that contains the full path of the DLL.

For example, the registry entry for a RAS administration DLL from a fictional company named Netwerks Corporation might be:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll
DisplayName : REG_SZ : Netwerks RAS Admin DLL
DLLPath : REG_SZ : C:\nt\system32\ntwkadm.dll

The setup program for a RAS administration DLL should also provide remove/uninstall functionality. If a user removes the DLL, the setup program should delete the DLL's registry entries.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RasAdminGetIpAddressForUser**

RAS Security DLL Functions

Implement the following functions when developing a RAS security DLL:

RasSecurityDialogBegin
RasSecurityDialogComplete
RasSecurityDialogEnd
RasSecurityDialogGetInfo
RasSecurityDialogReceive
RasSecurityDialogSend

RasSecurityDialogBegin

The **RasSecurityDialogBegin** function is a third-party RAS security DLL entry point that the Windows NT/Windows 2000 RAS server calls when a remote user tries to connect. This enables the security DLL to begin its authentication of the remote user.

Note that Windows NT/Windows 2000 currently provides RAS security host support only for serial devices; other types of connections, such as ISDN or a virtual private network (VPN) connection, are not supported.

```
DWORD WINAPI RasSecurityDialogBegin(
  HPORT hPort,        // RAS handle to the port
  PBYTE pSendBuf,     // pointer to buffer for sending data
  DWORD SendBufSize,  // size, in bytes, of the send buffer
  PBYTE pRecvBuf,     // pointer to buffer for receiving data
  DWORD RecvBufSize,  // size in bytes, of the receive buffer
  VOID (WINAPI * RasSecurityDialogComplete)
                      // pointer to the completion function
);
```

Parameters

hPort

Specifies a RAS port handle. The security DLL uses this handle in other RAS security functions, such as **RasSecurityDialogSend** and **RasSecurityDialogReceive**, to identify this authentication transaction.

Note that this handle is valid only in RAS security functions; you cannot use it in other Win32 I/O functions.

pSendBuf

Pointer to a buffer allocated by the RAS server. The security DLL uses this buffer with the **RasSecurityDialogSend** function to send text that is displayed in the RAS terminal window on the remote computer.

SendBufSize

Specifies the size, in bytes, of the *pSendBuf* buffer.

pRecvBuf

Pointer to a buffer allocated by the RAS server. The security DLL uses this buffer with the **RasSecurityDialogReceive** function to receive the response from the remote user.

RecvBufSize

Specifies the size, in bytes, of the *pRecvBuf* buffer.

RasSecurityDialogComplete

Specifies a pointer to a **RasSecurityDialogComplete** function. When the security DLL has completed the authentication of the remote user, it calls this function to report the results to the RAS server.

Return Values

If the security DLL successfully starts the authentication operation, **RasSecurityDialogBegin** should return NO_ERROR. In this case, the security DLL must later terminate the authentication transaction by calling the function pointed to by the *RasSecurityDialogComplete* parameter.

If an error occurs, **RasSecurityDialogBegin** should return a nonzero error code. In this case, the RAS server hangs up the call and records the error in the Windows NT/Windows 2000 event log. Returning a nonzero error code terminates the authentication transaction, so the security DLL does not need to call the *RasSecurityDialogComplete* function.

Remarks

When a Windows NT/Windows 2000 RAS server receives a call from a remote computer, it calls the **RasSecurityDialogBegin** function exported by the registered RAS security DLL, if there is one. When the RAS server calls this function, it passes the following information to the security DLL.

- A port handle to identify the connection
- Pointers to buffers to use when communicating with the remote user
- A pointer to a **RasSecurityDialogComplete** function to call when the authentication has been completed

The port handle and buffer pointers are valid until you call **RasSecurityDialogComplete** to terminate the authentication transaction.

Your **RasSecurityDialogBegin** implementation must return as soon as possible, because the RAS server is blocked and cannot accept any other calls until **RasSecurityDialogBegin** returns. The **RasSecurityDialogBegin** function should copy the input parameters and create a thread to communicate with and authenticate the remote user.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rasshost.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RasSecurityDialogComplete**, **RasSecurityDialogReceive**, **RasSecurityDialogSend**

RasSecurityDialogComplete

The **RasSecurityDialogComplete** function notifies the RAS server of the results of a third-party security authentication transaction. A third-party RAS security DLL calls **RasSecurityDialogComplete** when it has completed its authentication of the remote user.

The RAS server passes a pointer to the **RasSecurityDialogComplete** function when the server calls the **RasSecurityDialogBegin** entry point of the security DLL.

```
VOID RasSecurityDialogComplete(
  SECURITY_MESSAGE *pSecMsg  // pointer to the security
                             // message structure
);
```

Parameters

pSecMsg
Pointer to a **SECURITY_MESSAGE** structure that contains the results of the authentication transaction.

Return Values

None.

Remarks

When a security DLL has finished authenticating the remote user, it calls the **RasSecurityDialogComplete** function to report the results. The RAS server then performs a cleanup sequence. As part of this cleanup sequence, the RAS server calls the security DLL's **RasSecurityDialogEnd** function to give the DLL an opportunity to perform its own cleanup, if necessary.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasshost.h.
Library: Included as a resource in Rasman.dll.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RasSecurityDialogBegin**, **RasSecurityDialogComplete**, **RasSecurityDialogEnd**, **SECURITY_MESSAGE**

RasSecurityDialogEnd

The **RasSecurityDialogEnd** function is a third-party RAS security DLL entry point that the Windows NT/Windows 2000 RAS server calls to terminate an authentication transaction.

```
DWORD WINAPI RasSecurityDialogEnd(
  HPORT hPort  // RAS handle to the port
);
```

Parameters

hPort
Specifies the port handle that the RAS server passed to the security DLL in the **RasSecurityDialogBegin** call for this authentication transaction.

Return Values

If the security DLL returns NO_ERROR, the RAS server does not terminate the authentication transaction. In this case, the security DLL must later call the **RasSecurityDialogComplete** function when it is ready to terminate.

If the security DLL returns a nonzero error code, the RAS server terminates the authentication transaction. In this case, the security DLL does not need to make another **RasSecurityDialogComplete** call. You should return an error code defined in Winerror.h or Raserror.h, such as ERROR_PORT_DISCONNECTED.

Remarks

When a security DLL has finished authenticating the remote user, it calls the **RasSecurityDialogComplete** function. The RAS server then performs a cleanup sequence that includes a call to the DLL's **RasSecurityDialogEnd** function. This gives the security DLL an opportunity to perform any necessary cleanup. To terminate the authentication transaction, **RasSecurityDialogEnd** must return a nonzero error code.

The RAS server may also call **RasSecurityDialogEnd** if it needs to abnormally terminate the authentication transaction before the security DLL calls **RasSecurityDialogComplete**. In this case, the security DLL should terminate the worker thread associated with the *hPort* port handle, and perform any other necessary cleanup. If **RasSecurityDialogEnd** returns a nonzero value, the security DLL does not need to call **RasSecurityDialogComplete**.

For either a normal or abnormal termination, your **RasSecurityDialogEnd** function can return NO_ERROR to delay the termination. If it does so, it must later call **RasSecurityDialogComplete** when it is ready to terminate.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rasshost.h.

+ See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **RasSecurityDialogBegin**, **RasSecurityDialogComplete**

RasSecurityDialogGetInfo

The **RasSecurityDialogGetInfo** function is called by a RAS security DLL to get information about a port from the RAS server.

To call this function, you must first call the **LoadLibrary** function to load RASMAN.DLL. Then call the **GetProcAddress** function to get the DLL's **RasSecurityDialogGetInfo** entry point.

```
DWORD RasSecurityDialogGetInfo(
  HPORT hPort,  // RAS handle to port
  RAS_SECURITY_INFO *pBuffer
                // pointer to structure that gets port information
);
```

Parameters

hPort

Specifies the port handle that the RAS server passed to the security DLL in the **RasSecurityDialogBegin** call for this authentication transaction.

pBuffer

Pointer to a **RAS_SECURITY_INFO** structure that receives information about the specified RAS port.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is one of the error codes defined in Raserror.h or Winerror.h. **GetLastError** does not provide extended error information.

Remarks

The **RasSecurityDialogGetInfo** function retrieves information about the port associated with a RAS security DLL authentication transaction.

The **LastError** member of the **RAS_SECURITY_INFO** structure indicates the state of the last **RasSecurityDialogReceive** call for the port. If the receive operation has been completed successfully, **LastError** is SUCCESS and the **BytesReceived** member indicates the number of bytes received. Otherwise, **LastError** is PENDING if the receive operation is still in progress, or a nonzero error code if the receive operation failed.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasshost.h.
Library: Included as a resource in Rasman.dll.

+ See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **GetProcAddress**, **LoadLibrary**, **RAS_SECURITY_INFO**, **RasSecurityDialogReceive**

RasSecurityDialogReceive

The **RasSecurityDialogReceive** function starts an asynchronous operation that receives a remote user's response to a security challenge. The response is the input that the user typed in a terminal window on the remote computer. A third-party RAS security DLL calls this function as part of its authentication of the remote user.

To call this function, you must first call the **LoadLibrary** function to load RASMAN.DLL. Then call the **GetProcAddress** function to get the DLL's **RasSecurityDialogReceive** entry point.

```
DWORD WINAPI RasSecurityDialogReceive(
  HPORT hPort,    // RAS handle to the port
  PBYTE pBuffer,  // pointer to buffer that receives the
                  // user's response
  PWORD pBufferLength,
                  // returns size, in bytes, of the
                  // data received
  DWORD Timeout,  // time-out period, in seconds
  HANDLE hEvent   // event that is signaled when
                  // operation is finished
);
```

Parameters

hPort
Specifies the port handle that the RAS server passed to the security DLL in the **RasSecurityDialogBegin** call for this authentication transaction.

pBuffer
Pointer to the receive buffer that was passed to the security DLL in the **RasSecurityDialogBegin** call. When the asynchronous receive operation has been completed successfully, this buffer contains the response from the remote user.

pBufferLength
Pointer to a **WORD** variable. On input, this variable must specify the size, in bytes, of the *pBuffer* buffer. When the receive operation has been completed, the variable indicates the number of bytes returned in the *pBuffer* buffer.

Timeout
Specifies a time-out period, in seconds, after which the RAS server sets the *hEvent* event object to the signaled state.

If this value is zero, there is no time-out period; that is, the RAS server does not signal the event object until the receive operation has been completed.

hEvent
Specifies the handle of an event object created by the **CreateEvent** function. The RAS server sets the event object to the signaled state when the receive operation has been completed or when the time-out period has elapsed.

Return Values

If the function is successful, the return value is PENDING (defined in Raserror.h). This indicates that the receive operation is in progress.

If an error occurs, the return value is one of the error codes defined in Raserror.h or Winerror.h. **GetLastError** does not provide extended error information.

Remarks

After calling the **RasSecurityDialogSend** function to send a security challenge to the remote user, the security DLL must call the **RasSecurityDialogReceive** function to get the user's response.

The **RasSecurityDialogReceive** function is asynchronous. When the function returns, the security DLL must use one of the wait functions, such as **WaitForSingleObject**, to wait for the *hEvent* event object to be signaled. The RAS server signals the event object when the receive operation has been completed or when the time-out interval has elapsed. If the receive operation is successful, the *pBuffer* buffer contains the response from the remote user, and the *pBufferLength* parameter indicates the number of bytes received. If the remote user sends more bytes than will fit in the buffer, the RAS server buffers the excess bytes and returns them in the next **RasSecurityDialogReceive** call.

You can use the *Timeout* parameter to specify a time-out interval. If the time-out elapses, the RAS server signals the event object, and the *pBufferLength* parameter indicates that zero bytes were transferred. Alternatively, you can set *Timeout* to zero, and specify a time-out interval in the wait function that you use to wait for the event object to be signaled.

When a security DLL is authenticating a remote user, the connection operation on the remote computer enters a RASCS_Interactive paused state. The message sent by **RasSecurityDialogSend** is displayed as output in a terminal window on the remote computer. The response received by **RasSecurityDialogReceive** is the input that the remote user types in the terminal window. The RASCS_Interactive value is defined in the **RASCONNSTATE** enumeration.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasshost.h.
Library: Included as a resource in Rasman.dll.

+ See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **CreateEvent**, **GetProcAddress**, **LoadLibrary**, **RASCONNSTATE**, **RasSecurityDialogSend**, **WaitForSingleObject**

RasSecurityDialogSend

The **RasSecurityDialogSend** function sends a message to be displayed in a terminal window on a remote computer. A third-party RAS security DLL sends this message as part of its authentication of a remote user.

To call this function, you must first call the **LoadLibrary** function to load RASMAN.DLL. Then call the **GetProcAddress** function to get the DLL's **RasSecurityDialogSend** entry point.

```
DWORD RasSecurityDialogSend(
  HPORT hPort,          // RAS handle to the port
  PBYTE pBuffer,        // pointer to buffer containing
                        // data to send
  WORD BufferLength     // size, in bytes, of the data
                        // being sent
);
```

Parameters

hPort

Specifies the port handle that the RAS server passed to the security DLL in the **RasSecurityDialogBegin** call for this authentication transaction.

pBuffer

Pointer to the send buffer that was passed to the security DLL in the call to **RasSecurityDialogBegin**. Before calling **RasSecurityDialogSend**, copy into this buffer the message to send to the remote user. The *SendBufSize* parameter of the **RasSecurityDialogBegin** function indicates the maximum number of bytes you can copy to this buffer.

BufferLength

Specifies the number of bytes to send in the *pBuffer* buffer.

Return Values

If the function is successful, the return value is PENDING (defined in Raserror.h). This indicates that the send operation is in progress.

If an error occurs, the return value is one of the error codes defined in Raserror.h or Winerror.h. **GetLastError** does not provide extended error information.

Remarks

The **RasSecurityDialogSend** function is asynchronous. After calling it to send a message to the remote user, call the **RasSecurityDialogReceive** function, and then wait for a response. The security DLL can make any number of **RasSecurityDialogSend** calls, with each call followed by a **RasSecurityDialogReceive** call.

When a security DLL is authenticating a remote user, the connection operation on the remote computer enters a RASCS_Interactive paused state. The message sent by **RasSecurityDialogSend** is displayed as output in a terminal window on the remote computer. The response received by **RasSecurityDialogReceive** is the input that the remote user types in the terminal window. The RASCS_Interactive value is defined in the **RASCONNSTATE** enumeration.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasshost.h.
Library: Included as a resource in Rasman.dll.

+ See Also

Remote Access Service (RAS) Overview, RAS Server Administration Functions, **GetProcAddress**, **LoadLibrary**, **RASCONNSTATE**, **RasSecurityDialogBegin**, **RasSecurityDialogReceive**

RAS Server Administration Structures

For Windows NT version 4.0, use the following structures to implement RAS Server Administration functionality. Windows 95 does not provide RAS server support.

RAS_PARAMETERS
RAS_PORT_0
RAS_PORT_1
RAS_PORT_STATISTICS
RAS_PPP_ATCP_RESULT
RAS_PPP_IPCP_RESULT
RAS_PPP_IPXCP_RESULT
RAS_PPP_NBFCP_RESULT
RAS_PPP_PROJECTION_RESULT
RAS_SECURITY_INFO
RAS_SERVER_0
RAS_STATS
RAS_USER_0
SECURITY_MESSAGE

RAS_PARAMETERS

The **RAS_PARAMETERS** structure is used by the **RasAdminPortGetInfo** function to return the name and value of a media-specific parameter associated with a port on a Windows NT/Windows 2000 RAS Server.

```
struct RAS_PARAMETERS {
  CHAR  P_Key [RASSAPI_MAX_PARAM_KEY_SIZE];
  RAS_PARAMS_FORMAT  P_Type;
  BYTE  P_Attributes;
  RAS_PARAMS_VALUE  P_Value;
};
```

Members

P_Key

Specifies the name of the key that represents the media-specific parameter, such as MAXCONNECTBPS.

P_Type

Identifies the type of data associated with the parameter. This member can be one of the following values from the **RAS_PARAMS_FORMAT** enumeration.

Value	Meaning
ParamNumber	Indicates that the data associated with the key is a number.
ParamString	Indicates that the data associated with the key is a string.

P_Attributes

Reserved.

P_Value

Specifies the value associated with the parameter. This member is a **RAS_PARAMS_VALUE** union. If the **P_Type** member is ParamNumber, the **Number** member of the union contains the value. If **P_Type** is ParamString, the **String** member of the union contains the value.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RasAdminAcceptNewConnection**, **RasAdminConnectionHangupNotification**, **RasAdminPortGetInfo**

RAS_PORT_0

The **RAS_PORT_0** structure contains information that describes a RAS port.

```
typedef struct _RAS_PORT_0 {
  WCHAR wszPortName[RASSAPI_MAX_PORT_NAME];
  WCHAR wszDeviceType[RASSAPI_MAX_DEVICETYPE_NAME];
  WCHAR wszDeviceName[RASSAPI_MAX_DEVICE_NAME];
  WCHAR wszMediaName[RASSAPI_MAX_MEDIA_NAME];
  DWORD reserved;
  DWORD Flags;
  WCHAR wszUserName[UNLEN + 1];
  WCHAR wszComputer[NETBIOS_NAME_LEN];
  DWORD dwStartSessionTime;
  WCHAR wszLogonDomain[DNLEN + 1];
  BOOL fAdvancedServer;
} RAS_PORT_0, *PRAS_PORT_0;
```

Members

wszPortName

A null-terminated Unicode string that specifies the name of the port, such as "COM1".

wszDeviceType

A null-terminated Unicode string that specifies the type of the device on which the connection was made, such as "Modem" or "ISDN". The list of device types that might be specified in this member includes all the device types installed on the server, including third-party devices.

wszDeviceName

A null-terminated Unicode string that specifies the name of the device on which the connection was made, such as "Hayes 9600" or "PCIMACISDN1".

wszMediaName

A null-terminated Unicode string that specifies the name of the media used for the connection, such as "rasser" or "rastapi".

reserved

This member is reserved.

Flags

A set of bit flags that specify the nature of the connection made on this port. This member can be a combination of the following flags.

Value	Meaning
GATEWAY_ACTIVE	If this flag is set, the NetBIOS gateway is active on the server.
MESSENGER_PRESENT	If this flag is set, the Windows NT/Windows 2000 messenger service is running on the remote client.
PORT_MULTILINKED	If this flag is set, the port is multilinked with other ports. You can use this information for displaying the connection status as a multilinked port. For a multilinked port, the **RAS_PORT_STATISTICS** structure contains two sets of statistics: one for the port alone, and another for the combined ports in the multilink connection.
PPP_CLIENT	If this flag is set, the remote client connected using PPP. If this flag is not set, the remote client connected using the AMB protocol.
REMOTE_LISTEN	If this flag is set, the RemoteListen parameter of the NetBIOS gateway is set to 1 on the server.
USER_AUTHENTICATED	If this flag is set, a remote client is connected to the server and the user has been authenticated. You can check this flag to ensure that a client is actually connected to a port.

If the MESSENGER_PRESENT, GATEWAY_ACTIVE, and REMOTE_LISTEN flags are set, you can use the Windows NT/Windows 2000 messenger service to send an administrative message to the remote client. If MESSENGER_PRESENT and REMOTE_LISTEN are set, but GATEWAY_ACTIVE is not, you can send a message to the client only if you send the message from the RAS server the client is dialed in to.

wszUserName
A null-terminated Unicode string that specifies the name of the remote user connected to this port.

wszComputer
A null-terminated Unicode string that specifies the name of the remote client computer.

dwStartSessionTime
Specifies the time, in seconds from January 1, 1970, that the client connected to the RAS server on this port. You can use the standard Win32 time routines to format this value for display.

wszLogonDomain
A null-terminated Unicode string that specifies the name of the Windows NT/Windows 2000 domain on which the remote user was authenticated. This string is the domain name only, with no "\\" prefix.

fAdvancedServer
A flag that is nonzero if the RAS server associated with this port is a Windows NT/Windows 2000 Advanced Server. You can use this information to determine the name of the server that has the user account database. If the RAS server is an Advanced Server, you can get the name of the user account server by concatenating the prefix "\\" to the name returned in the **wszLogonDomain** member. This is because for an Advanced Server the local logon domain name is the same as the server name. If the RAS server is a Windows NT/Windows 2000 Workstation, you can use the **RasAdminGetUserAccountServer** function to get the name of the user account server.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RAS_PORT_1**, **RAS_PORT_STATISTICS**, **RasAdminGetUserAccountServer**, **RasAdminPortEnum**

RAS_PORT_1

The **RAS_PORT_1** structure contains information about a RAS port.

```
typedef struct _RAS_PORT_1 {
  RAS_PORT_0  rasport0;
  DWORD       LineCondition;
  DWORD       HardwareCondition;
  DWORD       LineSpeed;
  WORD        NumStatistics;
  WORD        NumMediaParms;
  DWORD       SizeMediaParms;
  RAS_PPP_PROJECTION_RESULT  ProjResult;
} RAS_PORT_1, *PRAS_PORT_1;
```

Members

rasPort0

A **RAS_PORT_0** structure that contains information about the port, such as the name of the port, the name of the remote user connected to the port, and so on.

LineCondition

Specifies the state of the port. This member can be one of the following values.

Value	Meaning
RAS_PORT_NON_OPERATIONAL	The port is not operational. Check the event log for errors reported by the server.
RAS_PORT_DISCONNECTED	The port is currently disconnected.
RAS_PORT_CALLING_BACK	The RAS server is calling back the RAS client.
RAS_PORT_LISTENING	The port is waiting for a client to call in.
RAS_PORT_AUTHENTICATING	The server is in the process of authenticating the remote client.
RAS_PORT_AUTHENTICATED	The remote client is now authenticated.
RAS_PORT_INITIALIZING	The device attached to the port is being initialized. The state of the port will change to RAS_PORT_LISTENING when the initialization has been completed.

HardwareCondition

Specifies one of the following values to indicate the state of the device attached to the port.

Value	Meaning
RAS_MODEM_OPERATIONAL	The modem attached to this port is operational and is ready to receive client calls.
RAS_MODEM_HARDWARE_FAILURE	The modem attached to this port has a hardware problem.

LineSpeed
Specifies the speed, in bits per second, with which the computer can communicate with the port.

NumStatistics
This member is not used. The RAS administration functions, such as the **RasAdminPortGetInfo** function, use the **RAS_PORT_STATISTICS** structure to return port statistics.

NumMediaParms
Specifies the number of media-specific parameters for this port. For serial media this is typically the number of values that appear in the SERIAL.INI file.

SizeMediaParms
Specifies the size, in bytes, of the buffer required for all media-specific parameters. The **RasAdminPortGetInfo** function returns a buffer containing an array of **RAS_PARAMETERS** structures with the media parameters and values for the port.

ProjResult
A **RAS_PPP_PROJECTION_RESULT** structure that specifies the PPP projection information for this port. This structure provides information for each protocol that is negotiated when a RAS client connects to a server.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RAS_PARAMETERS**, **RAS_PORT_0**, **RAS_PORT_STATISTICS**, **RAS_PPP_PROJECTION_RESULT**, **RasAdminAcceptNewConnection**, **RasAdminConnectionHangupNotification**, **RasAdminPortGetInfo**

RAS_PORT_STATISTICS

The **RAS_PORT_STATISTICS** structure reports the statistics that a RAS server collects for a connected port. The RAS server resets the various statistic counters each time the port is connected. You can call the **RasAdminPortClearStatistics** function to force the RAS server to reset the statistic counters.

For a port that is part of a multilink connection, this structure provides two sets of statistics. The first set contains the cumulative statistics for all ports in the connection. These statistics are the same for all ports in the connection. The second set contains the statistics for just this port. If the port is not part of a multilink connection, both sets of statistics have the same information. To determine whether a port is part of a multilink connection, check the PORT_MULTILINKED bit in the **Flags** member of the port's **RAS_PORT_0** structure.

```
typedef struct _RAS_PORT_STATISTICS
{
  // The connection statistics are followed by port statistics
  // A connection is across multiple ports.
  DWORD   dwBytesXmited;
  DWORD   dwBytesRcved;
  DWORD   dwFramesXmited;
  DWORD   dwFramesRcved;
  DWORD   dwCrcErr;
  DWORD   dwTimeoutErr;
  DWORD   dwAlignmentErr;
  DWORD   dwHardwareOverrunErr;
  DWORD   dwFramingErr;
  DWORD   dwBufferOverrunErr;
  DWORD   dwBytesXmitedUncompressed;
  DWORD   dwBytesRcvedUncompressed;
  DWORD   dwBytesXmitedCompressed;
  DWORD   dwBytesRcvedCompressed;

  // the following are the port statistics
  DWORD   dwPortBytesXmited;
  DWORD   dwPortBytesRcved;
  DWORD   dwPortFramesXmited;
  DWORD   dwPortFramesRcved;
  DWORD   dwPortCrcErr;
  DWORD   dwPortTimeoutErr;
  DWORD   dwPortAlignmentErr;
  DWORD   dwPortHardwareOverrunErr;
  DWORD   dwPortFramingErr;
  DWORD   dwPortBufferOverrunErr;
  DWORD   dwPortBytesXmitedUncompressed;
  DWORD   dwPortBytesRcvedUncompressed;
  DWORD   dwPortBytesXmitedCompressed;
  DWORD   dwPortBytesRcvedCompressed;

} RAS_PORT_STATISTICS, *PRAS_PORT_STATISTICS;
```

Members

dwBytesXmited
Specifies the total number of bytes transmitted by the connection.

dwBytesRcved
Specifies the total number of bytes received by the connection.

dwFramesXmited
Specifies the total number of frames transmitted by the connection.

dwFramesRcved
Specifies the total number of frames received by the connection.

dwCrcErr
Specifies the total number of CRC errors on the connection. CRC errors are caused by the failure of a cyclic redundancy check. A CRC error indicates that one or more characters in the data packet received were found garbled on arrival.

dwTimeoutErr
Specifies the total number of time-out errors on the connection. Time-out errors occur when an expected character is not received in time. When this occurs, the software assumes that the data has been lost and requests that it be resent.

dwAlignmentErr
Specifies the total number of alignment errors on the connection. Alignment errors occur when a character received is not the one expected. This usually happens when a character is lost or when a time-out error occurs.

dwHardwareOverrunErr
Specifies the total number of hardware overrun errors on the connection. These errors indicate the number of times the sending computer has transmitted characters faster than the receiving computer hardware can process them. If this problem persists, reduce the BPS connection rate on the client.

dwFramingErr
Specifies the total number of framing errors on the connection. A framing error occurs when an asynchronous character is received with an invalid start or stop bit.

dwBufferOverrunErr
Specifies the total number of buffer overrun errors on the connection. A buffer overrun error occurs when the sending computer is transmitting characters faster than the receiving computer can accommodate them. If this problem persists, reduce the BPS connection rate on the client.

dwBytesXmitedUncompressed
Specifies the total number of bytes transmitted uncompressed by the connection.

dwBytesRcvedUncompressed
Specifies the total number of bytes received uncompressed by the connection.

dwBytesXmitedCompressed
Specifies the total number of bytes transmitted compressed by the connection.

dwBytesRcvedCompressed
Specifies the total number of bytes received compressed by the connection.

dwPortBytesXmited
Specifies the total number of bytes transmitted by the port.

dwPortBytesRcved
Specifies the total number of bytes received by the port.

dwPortFramesXmited
Specifies the total number of frames transmitted by the port.

dwPortFramesRcved
Specifies the total number of frames received by the port.

dwPortCrcErr
Specifies the total number of CRC errors on the port. CRC errors are caused by the failure of a cyclic redundancy check. A CRC error indicates that one or more characters in the data packet received were found garbled on arrival.

dwPortTimeoutErr
Specifies the total number of time-out errors on the port. Time-out errors occur when an expected character is not received in time. When this occurs, the software assumes that the data has been lost and requests that it be resent.

dwPortAlignmentErr
Specifies the total number of alignment errors on the port. Alignment errors occur when a character received is not the one expected. This usually happens when a character is lost or when a time-out error occurs.

dwPortHardwareOverrunErr
Specifies the total number of hardware overrun errors on the port. These errors indicate the number of times the sending computer has transmitted characters faster than the receiving computer hardware can process them. If this problem persists, reduce the BPS connection rate on the client.

dwPortFramingErr
Specifies the total number of framing errors on the port. A framing error occurs when an asynchronous character is received with an invalid start or stop bit.

dwPortBufferOverrunErr
Specifies the total number of buffer overrun errors on the port. A buffer overrun error occurs when the sending computer is transmitting characters faster than the receiving computer can accommodate them. If this problem persists, reduce the BPS connection rate on the client.

dwPortBytesXmitedUncompressed
Specifies the total number of bytes transmitted uncompressed by the port. If the port is part of a multilink connection, this member is not valid. Use the compression statistics for the connection instead.

dwPortBytesRcvedUncompressed
Specifies the total number of bytes received uncompressed by the port. If the port is part of a multilink connection, this member is not valid. Use the compression statistics for the connection instead.

dwPortBytesXmitedCompressed
Specifies the total number of bytes transmitted compressed by the port. If the port is part of a multilink connection, this member is not valid. Use the compression statistics for the connection instead.

dwPortBytesRcvedCompressed
Specifies the total number of bytes received compressed by the port. If the port is part of a multilink connection, this member is not valid. Use the compression statistics for the connection instead.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RAS_PORT_0**, **RasAdminAcceptNewConnection**, **RasAdminConnectionHangupNotification**, **RasAdminPortClearStatistics**, **RasAdminPortGetInfo**

RAS_PPP_ATCP_RESULT

The **RAS_PPP_ATCP_RESULT** structure is used to report the result of an AppleTalk protocol projection operation for a port. Windows NT version 4.0 does not use this structure.

```
typedef struct _RAS_PPP_ATCP_RESULT {
  DWORD dwError;
  WCHAR wszAddress[ RAS_ATADDRESSLEN + 1 ];
} RAS_PPP_ATCP_RESULT;
```

Members

dwError
Specifies a value that indicates the results of the AppleTalk projection operation. A value of NO_ERROR indicates success, in which case, the **wszAddress** member is valid. If the projection operation is not successful, **dwError** is an error code from Winerror.h or Raserror.h.

wszAddress
Specifies a null-terminated Unicode string that specifies the IP address assigned to the remote client.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RAS_PPP_PROJECTION_RESULT**

RAS_PPP_IPCP_RESULT

The **RAS_PPP_IPCP_RESULT** structure is used to report the result of a PPP Internet Protocol (IP) projection operation for a port.

```
typedef struct _RAS_PPP_IPCP_RESULT {
  DWORD dwError;
  WCHAR wszAddress[ RAS_IPADDRESSLEN + 1 ];
} RAS_PPP_IPCP_RESULT;
```

Members

dwError

Indicates the results of the IP projection operation. A value of NO_ERROR indicates success, in which case, the **wszAddress** member is valid. If the projection operation was not successful, **dwError** is an error code from Winerror.h or Raserror.h.

wszAddress

A null-terminated Unicode string that specifies the IP address assigned to the remote client. This string has the "*a.b.c.d*" form.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RAS_PORT_1**, **RAS_PPP_PROJECTION_RESULT**, **RasAdminPortGetInfo**

RAS_PPP_IPXCP_RESULT

The **RAS_PPP_IPXCP_RESULT** structure is used to report the result of a PPP Internetwork Packet Exchange (IPX) projection operation for a port.

```
typedef struct _RAS_PPP_IPXCP_RESULT {
  DWORD dwError;
  WCHAR wszAddress[ RAS_IPXADDRESSLEN + 1 ];
} RAS_PPP_IPXCP_RESULT;
```

Members

dwError

Indicates the results of the IPX projection operation. A value of NO_ERROR indicates success, in which case, the **wszAddress** member is valid. If the projection operation was not successful, **dwError** is an error code from Winerror.h or Raserror.h.

wszAddress

A null-terminated Unicode string that specifies the IPX address assigned to the remote client. This string has the "*net.node*" form.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RAS_PORT_1**, **RAS_PPP_PROJECTION_RESULT**, **RasAdminPortGetInfo**

RAS_PPP_NBFCP_RESULT

The **RAS_PPP_NBFCP_RESULT** structure is used to report the result of a PPP NetBEUI Framer (NBF) projection operation for a port.

```
typedef struct _RAS_PPP_NBFCP_RESULT {
  DWORD dwError;
  DWORD dwNetBiosError;
  CHAR  szName[ NETBIOS_NAME_LEN + 1 ];
  WCHAR wszWksta[ NETBIOS_NAME_LEN + 1 ];
} RAS_PPP_NBFCP_RESULT;
```

Members

dwError

Indicates the results of the NBF projection operation. A value of NO_ERROR indicates success, in which case, the **wszWksta** member contains the name of the remote computer. If the projection operation was not successful, **dwError** is an error code from Winerror.h or Raserror.h.

dwNetBiosError

Ignore this member on the server; it is relevant only on the client.

szName

Ignore this member on the server; it is relevant only on the client.

wszWksta

A null-terminated Unicode string that specifies the NetBIOS name of the RAS client workstation.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RAS_PORT_1**, **RAS_PPP_PROJECTION_RESULT**, **RasAdminPortGetInfo**

RAS_PPP_PROJECTION_RESULT

The **RAS_PPP_PROJECTION_RESULT** structure is used to report the results of the various PPP projection operations for a port.

```
typedef struct _RAS_PPP_PROJECTION_RESULT {
  RAS_PPP_NBFCP_RESULT nbf;
  RAS_PPP_IPCP_RESULT  ip;
  RAS_PPP_IPXCP_RESULT ipx;
  RAS_PPP_ATCP_RESULT  at;
} RAS_PPP_PROJECTION_RESULT;
```

Members

nbf

A **RAS_PPP_NBFCP_RESULT** structure that reports the result of a PPP NetBEUI Framer (NBF) projection operation.

ip

A **RAS_PPP_IPCP_RESULT** structure that reports the result of a PPP Internet Protocol (IP) projection operation.

ipx

A **RAS_PPP_IPXCP_RESULT** structure that reports the result of a PPP Internetwork Packet Exchange (IPX) projection operation.

at

A **RAS_PPP_ATCP_RESULT** structure. Windows NT version 4.0 does not use this member.

Remarks

This structure reports the projection results for NetBEUI, TCP/IP, and IPX protocols. Each PPP structure has a **dwError** member that indicates whether the other information in the structure is valid. If **dwError** is NO_ERROR, the other information is valid. If **dwError** is one of the error codes in Winerror.h or Raserror.h, the other information is not valid.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RAS_PORT_1**, **RAS_PPP_ATCP_RESULT**, **RAS_PPP_IPCP_RESULT**, **RAS_PPP_IPXCP_RESULT**, **RAS_PPP_NBFCP_RESULT**, **RasAdminPortGetInfo**

RAS_SECURITY_INFO

The **RAS_SECURITY_INFO** structure is used with the **RasSecurityDialogGetInfo** function to return information about the RAS port associated with a RAS security DLL authentication transaction.

```
typedef struct _RAS_SECURITY_INFO {
  DWORD LastError;
  DWORD BytesReceived;
  CHAR DeviceName[RASSAPI_MAX_DEVICE_NAME+1];
}RAS_SECURITY_INFO,*PRAS_SECURITY_INFO;
```

Members

LastError

Specifies an error code that indicates the state of the last **RasSecurityDialogReceive** call for the port. If the receive operation failed, **LastError** is one of the error codes defined in Raserror.h or Winerror.h. Otherwise, **LastError** is one of the following values.

Value	Meaning
SUCCESS	The receive operation has been successfully completed. The **BytesReceived** member indicates the number of bytes received.
PENDING	The receive operation is pending completion.

BytesReceived
Specifies the number of bytes received in the most recent **RasSecurityDialogReceive** operation. This member is valid only if the value of the **LastError** member is SUCCESS.

DeviceName
Specifies a null-terminated string that contains the user-friendly display name of the device on the port, such as Hayes SmartModem 9600.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasshost.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RasSecurityDialogGetInfo**, **RasSecurityDialogReceive**

RAS_SERVER_0

The **RAS_SERVER_0** structure is used by the **RasAdminServerGetInfo** function to return information about the ports configured on a RAS Server.

```
typedef struct _RAS_SERVER_0 {
  WORD  TotalPorts;  // total ports available for connection
  WORD  PortsInUse;  // ports currently in use by
                     // remote clients
  DWORD RasVersion;  // version of RAS server
} RAS_SERVER_0, *PRAS_SERVER_0;
```

Members

TotalPorts
Specifies the total number of ports configured on the RAS server that are available for remote clients to connect to. For example, if the total number of ports configured for dialing in to a server is four, but one of the ports is currently in use for dialing out, **TotalPorts** will be three.

PortsInUse
Specifies the number of ports currently in use by remote clients.

RasVersion
Specifies the version of the RAS server. You can use this information to take version-specific action. This member can be one of the following values.

Value	Description
RASDOWNLEVEL	Indicates a LAN Manager version 1.0 RAS server.
RASADMIN_35	Indicates a Windows NT version 3.5 or 3.51 RAS server or client.
RASADMIN_CURRENT	Indicates a Windows NT version 4.0 RAS server or client.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RasAdminServerGetInfo**

RAS_STATS

The **RAS_STATS** structure stores the statistics for a single-link RAS connection, or for one of the links in a multilink RAS connection.

```
typedef struct _RAS_STATS {
  DWORD   dwSize;
  DWORD   dwBytesXmited;
  DWORD   dwBytesRcved;
  DWORD   dwFramesXmited;
  DWORD   dwFramesRcved;
  DWORD   dwCrcErr;
  DWORD   dwTimeoutErr;
  DWORD   dwAlignmentErr;
  DWORD   dwHardwareOverrunErr;
  DWORD   dwFramingErr;
  DWORD   dwBufferOverrunErr;
  DWORD   dwCompressionRatioIn;
  DWORD   dwCompressionRatioOut;
  DWORD   dwBps;
  DWORD   dwConnectDuration;
} RAS_STATS, *PRAS_STATS;
```

Members

dwSize

Specifies the version of the structure. Set this member to **sizeof(RAS_STATS)** before sing the structure in a function call.

dwBytesXmited
The number of bytes transmitted through this connection or link.

dwBytesRcved
The number of bytes received through this connection or link.

dwFramesXmited
The number frames transmitted through this connection or link.

dwFramesRcved
The number of frames received through this connection or link.

dwCrcErr
The number of Cyclic Redundancy Check (CRC) errors on this connection or link.

dwTimeoutErr
The number of timeout errors on this connection or link.

dwAlignmentErr
The number of alignment errors on this connection or link.

dwHardwareOverrunErr
The number of hardware overrun errors on this connection or link.

dwFramingErr
The number of framing errors on this connection or link.

dwBufferOverrunErr
The number of buffer overrun errors on this connection or link.

dwCompressionRatioIn
The compression ratio for the data being received on this connection or link.

dwCompressionRatioOut
The compression ratio for the data being transmitted on this connection or link.

dwBps
The speed of the connection or link, in bits per second.

For a single-link connection and for individual links in a multilink connection, this speed is negotiated at the time the connection or link is established.

For multilink connections, this speed is equal to the sum of the speeds of the individual links. For multilink connections, this speed will vary as links are added or deleted.

This speed is not equal to the throughput of the connection or link. To calculate the average throughput, divide the number of bytes transmitted (**dwBytesXmited**) and received (**dwBytesRcved**) by the amount of time the connection or link has been up (**dwConnectDuration**).

dwConnectDuration
The amount of time, in seconds, that the connection or link has been connected.

Requirements

Windows NT/2000: Requires Windows 2000.
Windows 95/98: Unsupported.
Header: Declared in Ras.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RasClearConnectionStatistics**, **RasClearLinkStatistics**, **RasGetConnectionStatistics**, **RasGetLinkStatistics**

RAS_USER_0

The **RAS_USER_0** structure is used in the **RasAdminUserSetInfo** and **RasAdminUserGetInfo** functions to specify information about a user.

```
typedef struct _RAS_USER_0 {
  BYTE  bfPrivilege;
  WCHAR szPhoneNumber[ RASSAPI_MAX_PHONENUMBER_SIZE + 1];
} RAS_USER_0, *PRAS_USER_0;
```

Members

bfPrivilege

A set of bit flags that specify the RAS privileges of the user. This member can be a combination of the RASPRIV_DialinPrivilege flag and one of the call-back flags. Note that when you call the **RasAdminUserSetInfo** function, you must specify one of the call-back flags. You can use the RASPRIV_CallbackType mask to identify the type of call-back privilege provided to the user. The following flags are defined.

Value	Meaning
RASPRIV_NoCallback	The user has no call-back privilege.
RASPRIV_AdminSetCallback	The user account is configured to have the administrator set the call-back number.
RASPRIV_CallerSetCallback	The remote user can specify a call-back phone number when dialing in.
RASPRIV_DialinPrivilege	The user has permission to dial in to this server.

szPhoneNumber

A null-terminated Unicode string that specifies the call-back phone number for the user.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RasAdminUserGetInfo**, **RasAdminUserSetInfo**

SECURITY_MESSAGE

The **SECURITY_MESSAGE** structure is used with the **RasSecurityDialogComplete** function to indicate the results of a RAS security DLL authentication transaction.

```
typedef struct _SECURITY_MESSAGE {
  DWORD dwMsgId;
  HPORT hPort;
  DWORD dwError;
  CHAR  UserName[UNLEN+1];
  CHAR  Domain[DNLEN+1];
} SECURITY_MESSAGE, *PSECURITY_MESSAGE;
```

Members

dwMsgId

Indicates whether the RAS server should grant access to the remote user. This member can be one of the following values.

Value	Meaning
SECURITYMSG_SUCCESS	The security DLL successfully authenticated the remote user identified by the **UserName** member. The RAS server will proceed with its PPP authentication.
SECURITYMSG_FAILURE	The security DLL denied access to the remote user identified by the **UserName** member. The RAS server will hang up the call and record the failed authentication in the Windows NT/2000 event log.
SECURITYMSG_ERROR	An error occurred that prevented validation of the remote user. The RAS server will hang up the call and record the error in the Windows NT/2000 event log.

hPort
Specifies the port handle that the RAS server passed to the security DLL in the **RasSecurityDialogBegin** call for this authentication transaction.

dwError
Specifies an error code. If **dwMsgId** is SECURITYMSG_ERROR, set **dwError** to one of the nonzero error codes defined in Winerror.h or Raserror.h. The RAS server records this error code in the Windows NT/Windows 2000 event log. If the **dwMsgId** member indicates success or failure, set **dwError** to zero.

UserName
Specifies the name of the remote user if **dwMsgId** is SECURITYMSG_SUCCESS or SECURITYMSG_FAILURE. This string can be empty if **dwMsgId** is SECURITYMSG_ERROR.

Domain
Specifies the name of the logon domain for the remote user if **dwMsgId** is SECURITYMSG_SUCCESS or SECURITYMSG_FAILURE. This string can be empty if **dwMsgId** is SECURITYMSG_ERROR.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Windows 95/98: Unsupported.
Header: Declared in Rasshost.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Structures, **RasSecurityDialogBegin**, **RasSecurityDialogComplete**

RAS Server Administration Union

For Windows NT version 4.0, use the following union to implement RAS Server Administration functionality. Windows 95 does not provide RAS server support.

RAS_PARAMS_VALUE

RAS_PARAMS_VALUE

The **RAS_PARAMS_VALUE** union is used in the **RAS_PARAMETERS** structure to store the data associated with a media-specific parameter. The **P_Type** member of the **RAS_PARAMETERS** structure uses a value from the **RAS_PARAMS_FORMAT** enumeration to indicate the type of value currently stored in **RAS_PARAMS_VALUE**.

```
union RAS_PARAMS_VALUE {
  DWORD  Number;
  struct  {
      DWORD  Length ;
      PCHAR  Data ;
  } String;
};
```

Members

Number

If the **P_Type** member of the **RAS_PARAMETERS** structure is ParamNumber, the **Number** member contains the value of the media-specific parameter. For example, the MAXCONNECTBPS parameter is of type ParamNumber, and the value might be 19200.

If the **P_Type** member of the **RAS_PARAMETERS** structure is ParamNumber, the **Number** member contains the value of the media-specific parameter. For example, the MAXCONNECTBPS parameter is of type ParamNumber, and the value might be 19200.

String

If the **P_Type** member of the **RAS_PARAMETERS** structure is ParamString, the **String** member contains the value of the media-specific parameter.

Length

Specifies the length, in characters, of the string pointed to by the **Data** member.

Data

Pointer to a buffer that contains the string value of a media-specific parameter.

! Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

+ See Also

Remote Access Service (RAS) Overview, RAS Server Administration Union, **RAS_PARAMETERS**, **RAS_PARAMS_FORMAT**

RAS Server Administration Enumeration Types

For Windows NT version 4.0, use the following enumeration to implement RAS Server Administration functionality. Windows 95 does not provide RAS server support.

RAS_PARAMS_FORMAT

RAS_PARAMS_FORMAT

The **RAS_PARAMS_FORMAT** enumeration type is used in the **RAS_PARAMETERS** structure to indicate the type of data associated with a media-specific key.

```
enum RAS_PARAMS_FORMAT {
  ParamNumber        = 0,
  ParamString        = 1
} ;
```

Enumerator Value	Meaning
ParamNumber	Indicates that the data associated with the key is a number.
ParamString	Indicates that the data associated with the key is a string.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service (RAS) Overview, RAS Server Administration Enumeration Types, **RAS_PARAMETERS**

CHAPTER 11

RRAS Overview

About Routing and Remote Access Service

The following chapters describe the API for the Routing and Remote Access Service (RRAS). RRAS is a feature of Microsoft® Windows® 2000.

The RRAS API has the following components:

- RAS Administration
- Router Administration
- Routing Protocol Interface
- Routing Table Manager Version 1
- Routing Table Manager Version 2
- Extensible Authentication Protocol
- Tracing

Windows 2000 RRAS Registry Layout

The following syntax shows an example registry layout for the router service.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\RasMan
    \PPP
        \ControlProtocols
            \Builtin
                Path: REG_EXPAND_SZ: %SystemRoot%\System32\rasppp.dll
            \Chap
                Path: REG_EXPAND_SZ: %SystemRoot%\System32\raschap.dll
        \EAP
            \<typeID>
                ConfigCLSID: REG_SZ: <guid>
                ConfigUiPath: REG_EXPAND_SZ: %SystemRoot%\System32\rastls.dll
                FriendlyName: REG_SZ: Public Key Based Authentication (EAP-TLS)
                IdentityUIPath: REG_EXPAND_SZ: %SystemRoot%\System32\rastls.dll
                InvokePasswordDialog: REG_DWORD: 0
                InvokeUsernameDialog: REG_DWORD: 0
                Path: REG_EXPAND_SZ: %SystemRoot%\System32\rastls.dll
            \<typeID>
```

(continued)

(continued)

```
                FriendlyName: REG_SZ: MD5 CHAP
                InvokePasswordDialog: REG_DWORD: 0x1
                InvokeUsernameDialog: REG_DWORD: 0x1
                Path: REG_EXPAND_SZ: %SystemRoot%\System32\raschap.dll
                StandaloneSupported: REG_DWORD: 0

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\RemoteAccess
    \Accounting
        \Providers
            ActiveProvider: REG_SZ: . . .
            \<guid>
                ConfigCLSID: REG_SZ: <guid>
                DisplayName: REG_SZ: Radius Accounting
                Path: REG_EXPAND_SZ: %SystemRoot%\System32\rasrad.dll
                Vendor: REG_SZ: Microsoft
             . . .
    \Authentication
        \Providers
            ActiveProvider: REG_SZ: . . .
            \<guid>
                ConfigCLSID: REG_SZ: <guid>
                DisplayName: REG_SZ: Radius Authentication
                Path: REG_EXPAND_SZ: %SystemRoot%\System32\rasrad.dll
                Vendor: REG_SZ: Microsoft
            \<guid>
                ConfigCLSID: REG_SZ: <guid>
                DisplayName: REG_SZ: NT Authentication
                Path: REG_EXPAND_SZ: %SystemRoot%\System32\rasauth.dll
                Vendor: REG_SZ: Microsoft
             . . .
    \DemandDialManager
        DLLPath: REG_EXPAND_SZ: . . .
        < RAS parameters and DDM parameters >
    \Interfaces
        \0
            Enabled: REG_DWORD: (0/1)
            InterfaceName: REG_SZ: Redmond
            Type: REG_DWORD: (Internal/Dedicated/Loopback)
                \IP
                    InterfaceInfo: REG_BINARY: . . .
                    ProtocolID: REG_DWORD: 0x0021
                \IPX
                    InterfaceInfo: REG_BINARY: . . .
                    ProtocolID: REG_DWORD: 0x002B
```

```
            . . .
        \N
            InterfaceName: REG_SZ: IntelEtherExpressPro2
            . . .
    \Parameters
        LANOnlyMode: REG_DWORD: (0/1)
        ServerFlags: REG_DWORD: . . .
        ServiceDLL: REG_EXPAND_SZ: %SystemRoot%\System32\mprdim.dll
    \RouterManagers
        \IP
            DLLPath: REG_SZ: . . .
            GlobalInFilter: REG_BSZ: < filter set name > . . .
            GlobalInfo: REG_BINARY: . . .
            GlobalInterfaceInfo: REG_BINARY: . . .
            ProtocolID: REG_DWORD: 0x0021
            . . .
            \IPX
            DLLPath: REG_SZ: . . .
            GlobalInFilter: REG_BSZ: < filter set name > . . .
            GlobalInfo: REG_BINARY: . . .
            GlobalInterfaceInfo: REG_BINARY: . . .
            ProtocolID: REG_DWORD: 0x002B
            . . .
        . . .

HKEY_LOCAL_MACHINE\Software\Microsoft
    \Router
        \CurrentVersion
            \RouterManagers
                \IP
                    \OSPF
                        ConfigDll: REG_SZ: ipadmin.dll
                        DllName: REG_SZ: ospf.dll
                        ProtocolID: REG_DWORD: 0xD
                        Title: REG_SZ: Open Shortest Path First
                    \IPBOOTP
                        . . .
                    \IPRIP
                        . . .
                \IPX
                    \IpxRip
                        ConfigDll: REG_SZ: ipxadmin.dll
                        DllName: REG_SZ: IPXRIP.DLL
                        ProtocolID: REG_DWORD: 0x20000
```

(continued)

(continued)

```
                    Title: REG_SZ: RIP for Ipx
                \IpxSap
                    . . .
                . . .
        \UIConfigDlls
            <guid1>: REG_SZ: ifadmin.dll
            <guid2>: REG_SZ: ipadmin.dll
            <guid3>: REG_SZ: ipxadmin.dll
            <guid4>: REG_SZ: ddmadmin.dll
```

Every router manager installed in the system will have a registry key created under the Router key. The DLLPath variable specifies the location of the DLL corresponding to the router manager and the ProtocolID variable specifies the protocol family identifier for the router manager.

The Interfaces key is populated with the interfaces that have been added to the local system from the Router configuration. Each interface has an associated Type (Internal, Dedicated, or Dynamic) and subkeys for each router manager (IP and IPX for example).

About Remote Access Service Administration

Microsoft® Windows® 2000 provides a set of functions for administering user permissions and ports on Windows 2000 RAS servers. Using these functions, you can develop a RAS server administration application to perform the following tasks:

- Enumerate those users who have a specified set of RAS permissions
- Assign or revoke RAS permissions for a specified user
- Enumerate the configured ports on a RAS server
- Get information and statistics about a specified port on a RAS server
- Reset the statistics counters for a specified port
- Disconnect a specified port

You can also install a RAS server administration DLL for auditing user connections and assigning IP addresses to dial-in users. The DLL exports a set of functions that the RAS server calls whenever a user tries to connect or disconnect.

RAS User Administration

A Windows® 2000 RAS server uses a user account database that contains information about a set of user accounts. The information includes a user's RAS privileges, which are a set of bit flags that determine how the RAS server responds when the user calls to connect. You can use the RAS server administration functions to locate the user account database, and to get and set the RAS privileges for user accounts.

A Windows 2000 RAS server can be part of a Windows 2000 domain, or it can be a stand-alone Windows 2000 Server or Windows 2000 Professional workstation that is not part of a domain. For a server that is part of a domain, the user account database is stored on the Windows NT/Windows 2000 server that is the Primary Domain Controller (PDC). A stand-alone server stores its own local user account database. To get the name of the server that stores the user account database used by a specified RAS server, you can call the **MprAdminGetPDCServer** function. You can then use the name of the user account server in a call to the **NetQueryDisplayInformation** function to enumerate the users in a user account database. You can also use the server name in calls to the **MprAdminUserGetInfo** and **MprAdminUserSetInfo** functions to get and set the RAS privileges for a specified user account.

The **MprAdminUserGetInfo** and **MprAdminUserSetInfo** functions use the **RAS_USER_0** structure to specify a user's RAS privileges and call-back phone number. The RAS privileges indicate the following information:

- Whether the user can make a remote connection to the server or the domain to which the server belongs.
- Whether the user can establish a connection through a call back, in which the RAS server hangs up and then calls back to the user to establish the connection.

Each user account specifies one of the following flags to indicate the user's call-back privileges.

Value	Meaning
RASPRIV_NoCallback	The RAS server will not call back the user to establish a connection.
RASPRIV_AdminSetCallback	When the user calls, the RAS server hangs up and calls a preset call-back phone number stored in the user account database. The **szPhoneNumber** member of the **RAS_USER_0** structure contains the user's call-back phone number.
RASPRIV_CallerSetCallback	When the user calls, the RAS server provides the option of specifying a phone number to call back. The user can also choose to connect immediately without a call back. The **szPhoneNumber** member contains a default number that the user can override.

RAS Server and Port Administration

You can use the RAS server administration functions to get information about a specified RAS server and its ports. These functions can also be used to terminate a connection on a specified RAS server port.

The **MprAdminServerGetInfo** function returns a **MPR_SERVER_0** structure that contains information about the configuration of a RAS server. The returned information includes the number of ports currently available for connection, the number of ports currently in use, and the server version number.

The **MprAdminPortEnum** function retrieves an array of **RAS_PORT_0** structures that contains information for each of the ports configured on a RAS server. The information for each port includes:

- The name of the port
- Information about the device attached to the port
- Whether the RAS server associated with the port is a Windows NT/Windows 2000 Server
- Whether the port is currently in use, and, if it is, information about the connection

You can call the **MprAdminPortGetInfo** function to get additional information about a specified port on a RAS server. This function returns a **RAS_PORT_1** structure that contains a **RAS_PORT_0** structure and additional information about the current state of the port. The **RasAdminPortGetInfo** function also returns an array of **RAS_PARAMETERS** structures that describe the values of any media-specific keys associated with the port. A **RAS_PARAMETERS** structure uses a value from the **RAS_PARAMS_FORMAT** enumeration to indicate the format of the value for each media-specific key.

The **MprAdminPortGetInfo** function also returns a **RAS_PORT_STATISTICS** structure that contains various statistic counters for the current connection, if any, on the port. For a port that is part of a multilink connection, **MprAdminPortGetInfo** returns statistics for the individual port and cumulative statistics for all ports involved in the connection. You can use the **MprAdminPortClearStats** function to reset the statistic counters for the port. The **MprAdminPortDisconnect** function disconnects a port that is in use.

Use the **MprAdminBufferFree** function to free memory allocated by the **MprAdminPortEnum** and **MprAdminPortGetInfo** functions. Use the **MprAdminGetErrorString** function to get a string that describes a RAS error code returned by one of the RAS server administration (RasAdmin) functions.

RAS Administration DLL

Microsoft® Windows NT® version 4.0 makes it possible for you to install a RAS administration DLL on a Windows NT version 4.0 RAS server. The DLL exports functions that the RAS server calls whenever a user tries to connect or disconnect. You can use the DLL to perform the following administrative functions:

- Decide whether to allow a user to connect to the server. This can provide a security check in addition to the standard RAS user authentication.
- Record the time that each user connects to and disconnects from the server. This can be useful for billing or auditing purposes.

- Assign an IP address to each user. This can be useful for security, since you can use this feature to map a user's connection to a specific computer.

Implement the following functions when developing a RAS server administration DLL:

- **MprAdminAcceptNewConnection**
- **MprAdminConnectionHangupNotification**
- **MprAdminGetIpAddressForUser**
- **MprAdminReleaseIpAddress**

A RAS administration DLL must implement and export all of the above functions. If any of the functions are not implemented, the remote access service will not start.

The **MprAdminAcceptNewConnection** and **MprAdminConnectionHangupNotification** functions enable the DLL to audit user connections to the server. A Windows NT/Windows 2000 RAS server calls the **MprAdminAcceptNewConnection** function whenever a user tries to connect. This function can prevent the user from connecting. You can also use the **MprAdminAcceptNewConnection** function to generate an entry in a log for billing or auditing. When the user disconnects, the RAS server calls the **MprAdminConnectionHangupNotification** function, which can log the time at which the user disconnected.

After the RAS server has authenticated a caller, it calls the **MprAdminGetIpAddressForUser** function to get an IP address for the remote client. The DLL can use this function to provide an alternate scheme to map an IP address to a dial-in user. If **MprAdminGetIpAddressForUser** is not implemented, a RAS server connects a remote user to an IP address that is selected from a static pool of IP addresses, or one selected by a Dynamic Host Configuration Protocol (DHCP) server. The **MprAdminGetIpAddressForUser** function allows the DLL to override this default IP address and specify a particular IP address for each user. The **MprAdminGetIpAddressForUser** function can set a flag that causes RAS to call the **MprAdminReleaseIPAddress** function when the user disconnects. The DLL can use **MprAdminReleaseIPAddress** to update its user-to-IP-address map.

RAS serializes calls into the administration DLL. A call into one of the DLL's functions for a given RAS client will not be preempted by a call to that function for a different RAS client; RAS will not call the function for the other client until the initial call is complete. Furthermore, serialization extends to certain groups of functions. The IP address functions are serialized as a group; a call into either **MprAdminGetIpAddressForUser** or **MprAdminReleaseIpAddress** blocks calls into both functions until the initial call is complete. **MprAdminAcceptNewConnection** and **MprAdminConnectionHangupNotification** are also serialized as a group.

RAS executes the functions for assigning IP addresses in one process; the functions for connection and disconnection notifications are executed in another process. Consequently, the DLL should not depend on shared data between these two sets of functions.

The RAS server logs an error in the system event log if an error occurs when it tries to load a RAS administration DLL or when calling one of the DLL's functions. This can happen, for example, if the DLL specified the wrong name for an exported function, or if it did not include the function name in the DEF file. The entry in the event log indicates the reason for the failure.

Windows 2000 and later: RAS administration DLLs that implement this function interface do not work on Windows 2000 and later versions. For Windows 2000 and later versions, use the MprAdmin function interface provided with the more recent versions of Windows. For more information, see the RAS Administration Reference in the Routing and RAS documentation.

RAS Administration DLL Registry Setup

The setup program for a third-party RAS administration DLL must register the DLL with RAS by providing information under the following key in the registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll

To register the DLL, set the following values under this key.

Value name	Value data
DisplayName	A REG_SZ string that contains the user-friendly display name of the DLL.
DLLPath	A REG_SZ string that contains the full path of the DLL.

For example, the registry entry for a RAS administration DLL from a fictional company named Netwerks Corporation might be:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll
DisplayName : REG_SZ : Netwerks RAS Admin DLL
DLLPath : REG_SZ : C:\nt\system32\ntwkadm.dll

The setup program for a RAS administration DLL should also provide remove/uninstall functionality. If a user removes the DLL, the setup program should delete the registry entries for the DLL.

CHAPTER 12

Remote Access Service Administration

Remote Access Services Administration Overview

Microsoft® Windows® 2000 provides a set of functions for administering user permissions and ports on Windows 2000 RAS servers. Using these functions, you can develop a RAS server administration application to perform the following tasks:

- Enumerate those users who have a specified set of RAS permissions
- Assign or revoke RAS permissions for a specified user
- Enumerate the configured ports on a RAS server
- Get information and statistics about a specified port on a RAS server
- Reset the statistics counters for a specified port
- Disconnect a specified port

You can also install a RAS server administration DLL for auditing user connections and assigning IP addresses to dial-in users. The DLL exports a set of functions that the RAS server calls whenever a user tries to connect or disconnect.

RAS User Administration

A Windows® 2000 RAS server uses a user account database that contains information about a set of user accounts. The information includes a user's RAS privileges, which are a set of bit flags that determine how the RAS server responds when the user calls to connect. You can use the RAS server administration functions to locate the user account database, and to get and set the RAS privileges for user accounts.

A Windows 2000 RAS server can be part of a Windows 2000 domain, or it can be a stand-alone Windows 2000 Server or Windows 2000 Professional workstation that is not part of a domain. For a server that is part of a domain, the user account database is stored on the Windows NT/Windows 2000 server that is the Primary Domain Controller (PDC). A stand-alone server stores its own local user account database. To get the name of the server that stores the user account database used by a specified RAS server, you can call the **MprAdminGetPDCServer** function. You can then use the name of the user account server in a call to the **NetQueryDisplayInformation** function to enumerate the users in a user account database. You can also use the server name in calls to the **MprAdminUserGetInfo** and **MprAdminUserSetInfo** functions to get and set the RAS privileges for a specified user account.

The **MprAdminUserGetInfo** and **MprAdminUserSetInfo** functions use the **RAS_USER_0** structure to specify a user's RAS privileges and call-back phone number. The RAS privileges indicate the following information:

- Whether the user can make a remote connection to the server or the domain to which the server belongs.
- Whether the user can establish a connection through a call back, in which the RAS server hangs up and then calls back to the user to establish the connection.

Each user account specifies one of the following flags to indicate the user's call-back privileges.

Value	Meaning
RASPRIV_NoCallback	The RAS server will not call back the user to establish a connection.
RASPRIV_AdminSetCallback	When the user calls, the RAS server hangs up and calls a preset call-back phone number stored in the user account database. The **szPhoneNumber** member of the **RAS_USER_0** structure contains the user's call-back phone number.
RASPRIV_CallerSetCallback	When the user calls, the RAS server provides the option of specifying a phone number to call back. The user can also choose to connect immediately without a call back. The **szPhoneNumber** member contains a default number that the user can override.

RAS Server and Port Administration

You can use the RAS server administration functions to get information about a specified RAS server and its ports. These functions can also be used to terminate a connection on a specified RAS server port.

The **MprAdminServerGetInfo** function returns a **MPR_SERVER_0** structure that contains information about the configuration of a RAS server. The returned information includes the number of ports currently available for connection, the number of ports currently in use, and the server version number.

The **MprAdminPortEnum** function retrieves an array of **RAS_PORT_0** structures that contains information for each of the ports configured on a RAS server. The information for each port includes:

- The name of the port
- Information about the device attached to the port

- Whether the RAS server associated with the port is a Windows NT/Windows 2000 Server
- Whether the port is currently in use, and, if it is, information about the connection

You can call the **MprAdminPortGetInfo** function to get additional information about a specified port on a RAS server. This function returns a **RAS_PORT_1** structure that contains a **RAS_PORT_0** structure and additional information about the current state of the port. The **RasAdminPortGetInfo** function also returns an array of **RAS_PARAMETERS** structures that describe the values of any media-specific keys associated with the port. A **RAS_PARAMETERS** structure uses a value from the **RAS_PARAMS_FORMAT** enumeration to indicate the format of the value for each media-specific key.

The **MprAdminPortGetInfo** function also returns a **RAS_PORT_STATISTICS** structure that contains various statistic counters for the current connection, if any, on the port. For a port that is part of a multilink connection, **MprAdminPortGetInfo** returns statistics for the individual port and cumulative statistics for all ports involved in the connection. You can use the **MprAdminPortClearStats** function to reset the statistic counters for the port. The **MprAdminPortDisconnect** function disconnects a port that is in use.

Use the **MprAdminBufferFree** function to free memory allocated by the **MprAdminPortEnum** and **MprAdminPortGetInfo** functions. Use the **MprAdminGetErrorString** function to get a string that describes a RAS error code returned by one of the RAS server administration (RasAdmin) functions.

RAS Administration DLL

Microsoft® Windows NT® version 4.0 makes it possible for you to install a RAS administration DLL on a Windows NT version 4.0 RAS server. The DLL exports functions that the RAS server calls whenever a user tries to connect or disconnect. You can use the DLL to perform the following administrative functions:

- Decide whether to allow a user to connect to the server. This can provide a security check in addition to the standard RAS user authentication.
- Record the time that each user connects to and disconnects from the server. This can be useful for billing or auditing purposes.
- Assign an IP address to each user. This can be useful for security, since you can use this feature to map a user's connection to a specific computer.

Implement the following functions when developing a RAS server administration DLL:

- **MprAdminAcceptNewConnection**
- **MprAdminConnectionHangupNotification**
- **MprAdminGetIpAddressForUser**
- **MprAdminReleaseIpAddress**

A RAS administration DLL must implement and export all of the above functions. If any of the functions are not implemented, the remote access service will not start.

The **MprAdminAcceptNewConnection** and **MprAdminConnectionHangupNotification** functions enable the DLL to audit user connections to the server. A Windows NT/Windows 2000 RAS server calls the **MprAdminAcceptNewConnection** function whenever a user tries to connect. This function can prevent the user from connecting. You can also use the **MprAdminAcceptNewConnection** function to generate an entry in a log for billing or auditing. When the user disconnects, the RAS server calls the **MprAdminConnectionHangupNotification** function, which can log the time at which the user disconnected.

After the RAS server has authenticated a caller, it calls the **MprAdminGetIpAddressForUser** function to get an IP address for the remote client. The DLL can use this function to provide an alternate scheme to map an IP address to a dial-in user. If **MprAdminGetIpAddressForUser** is not implemented, a RAS server connects a remote user to an IP address that is selected from a static pool of IP addresses, or one selected by a Dynamic Host Configuration Protocol (DHCP) server. The **MprAdminGetIpAddressForUser** function allows the DLL to override this default IP address and specify a particular IP address for each user. The **MprAdminGetIpAddressForUser** function can set a flag that causes RAS to call the **MprAdminReleaseIPAddress** function when the user disconnects. The DLL can use **MprAdminReleaseIPAddress** to update its user-to-IP-address map.

RAS serializes calls into the administration DLL. A call into one of the DLL's functions for a given RAS client will not be preempted by a call to that function for a different RAS client; RAS will not call the function for the other client until the initial call is complete. Furthermore, serialization extends to certain groups of functions. The IP address functions are serialized as a group; a call into either **MprAdminGetIpAddressForUser** or **MprAdminReleaseIpAddress** blocks calls into both functions until the initial call is complete. **MprAdminAcceptNewConnection** and **MprAdminConnectionHangupNotification** are also serialized as a group.

RAS executes the functions for assigning IP addresses in one process; the functions for connection and disconnection notifications are executed in another process. Consequently, the DLL should not depend on shared data between these two sets of functions.

The RAS server logs an error in the system event log if an error occurs when it tries to load a RAS administration DLL or when calling one of the DLL's functions. This can happen, for example, if the DLL specified the wrong name for an exported function, or if it did not include the function name in the DEF file. The entry in the event log indicates the reason for the failure.

Windows 2000 and later: RAS administration DLLs that implement this function interface do not work on Windows 2000 and later versions. For Windows 2000 and later versions, use the MprAdmin function interface provided with the more recent versions of Windows. For more information, see the RAS Administration Reference in the Routing and RAS documentation.

RAS Administration DLL Registry Setup

The setup program for a third-party RAS administration DLL must register the DLL with RAS by providing information under the following key in the registry:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll

To register the DLL, set the following values under this key.

Value name	Value data
DisplayName	A REG_SZ string that contains the user-friendly display name of the DLL.
DLLPath	A REG_SZ string that contains the full path of the DLL.

For example, the registry entry for a RAS administration DLL from a fictional company named Netwerks Corporation might be:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS\AdminDll
DisplayName : REG_SZ : Netwerks RAS Admin DLL
DLLPath : REG_SZ : C:\nt\system32\ntwkadm.dll

The setup program for a RAS administration DLL should also provide remove/uninstall functionality. If a user removes the DLL, the setup program should delete the registry entries for the DLL.

Remote Access Service Administration Reference

This chapter describes the reference elements used to implement the Remote Access Service (RAS) for Microsoft® Windows NT® version 4.0.

The RAS API is distributed as a feature of Microsoft Windows 2000. RAS can also be downloaded and used as a component of either Windows 2000 or Windows NT 4.0. RAS in either of these forms provides the same functionality. The only difference is the naming convention that is used for the reference elements in each version of the RAS API.

The functions that are used to implement RAS for Windows NT 4.0 typically begin with the "RasAdmin" prefix. The analogous functions for RRAS begin with the "MprAdmin" prefix.

For example, Windows NT 4.0 RAS provides a function called **RasAdminPortGetInfo**. The analogous function in RRAS is called **MprAdminPortGetInfo**. Another example: Windows NT 4.0 RAS provides the callback function **RasAdminGetIpAddressForUser**. RRAS provides a similar callback function called **MprAdminGetIpAddressForUser**. Exceptions to this rule are **RasAdminPortClearStatistics**, which, under RRAS is **MprAdminPortClearStats**, and **RasAdminFreeBuffer**, which under RRAS is **MprAdminBufferFree**.

The following table lists the Windows NT 4.0 RAS functions and the corresponding RRAS functions.

Windows NT 4.0 RAS	RRAS
RasAdminAcceptNewConnection	**MprAdminAcceptNewConnection**
RasAdminConnectionHangupNotification	**MprAdminConnectionHangupNotification**
RasAdminFreeBuffer	**MprAdminBufferFree**
RasAdminGetErrorString	**MprAdminGetErrorString**
RasAdminGetIpAddressForUser	**MprAdminGetIpAddressForUser**
RasAdminPortClearStatistics	**MprAdminPortClearStats**
RasAdminPortDisconnect	**MprAdminPortDisconnect**
RasAdminPortEnum	**MprAdminPortEnum**
RasAdminPortGetInfo	**MprAdminPortGetInfo**
RasAdminReleaseIpAddress	**MprAdminReleaseIpAddress**
RasAdminUserGetInfo	**MprAdminUserGetInfo**
RasAdminUserSetInfo	**MprAdminUserSetInfo**

Although the RRAS functions are similar to their Windows NT 4.0 RAS counterparts in functionality, RRAS functions often take a different set of parameters. See the reference page for a particular function for complete information on that function's parameter list.

The RRAS redistributable for Windows NT 4.0 adds the following functions, which have no counterparts in Windows NT 4.0 RAS:

MprAdminAcceptNewLink
MprAdminConnectionClearStats
MprAdminConnectionEnum
MprAdminConnectionGetInfo
MprAdminGetPDCServer
MprAdminIsServiceRunning
MprAdminLinkHangupNotification
MprAdminPortReset
MprAdminServerConnect
MprAdminServerDisconnect

In addition to the preceding functions, Windows 2000 adds the following functions:

MprAdminSendUserMessage
MprAdminAcceptNewConnection2
MprAdminConnectionHangupNotification2

RAS Administration Functions

This documentation describes RRAS functions that are used to develop software to administer RAS dial-up connections. These functions include:

MprAdminConnectionClearStats
MprAdminConnectionEnum
MprAdminConnectionGetInfo
MprAdminPortClearStats
MprAdminPortDisconnect
MprAdminPortEnum
MprAdminPortGetInfo
MprAdminPortReset

Additional functions are used for both RAS administration and router administration. These functions are listed following and are documented in the Router Administration Functions reference:

MprAdminBufferFree
MprAdminGetErrorString
MprAdminIsServiceRunning
MprAdminServerConnect
MprAdminServerDisconnect

MprAdminConnectionClearStats

The **MprAdminConnectionClearStats** function resets the statistics counters for the specified connection.

```
DWORD MprAdminConnectionClearStats(
  RAS_SERVER_HANDLE hRasServer,   // handle to server
  HANDLE hRasConnection           // handle to the connection
);
```

Parameters

hRasServer

Handle to the Remote Access Server on which to execute **MprAdminConnectionClearStats**. Obtain this handle by calling **MprAdminServerConnect**.

hRasConnection

Handle to the connection for which to reset the statistics. Obtain this handle by calling **MprAdminConnectionEnum**.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is ERROR_INVALID_PARAMETER.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminConnectionClearStats** rather than **MprAdminConnectionClearStats**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminConnectionEnum**, **MprAdminServerConnect**

MprAdminConnectionEnum

The **MprAdminConnectionEnum** function enumerates all active connections.

```
DWORD MprAdminConnectionEnum(
  RAS_SERVER_HANDLE hRasServer, // handle to the server
  DWORD dwLevel,                // must be zero, one, or two
  LPBYTE *lplpbBuffer,          // pointer to array of
                                // connection structs
  DWORD dwPrefMaxLen,           // maximum preferred length
```

```
                                // of returned data
  LPDWORD lpdwEntriesRead,      // number of connections
                                // enumerated
  LPDWORD lpdwTotalEntries,     // number of connections
                                // that could've been
                                // enumerated
  LPDWORD lpdwResumeHandle      // handle with which to
                                // resume enumeration
);
```

Parameters

hRasServer
Handle to the Remote Access Server on which connections are enumerated. Obtain this handle by calling **MprAdminServerConnect**.

dwLevel
Specifies the format of the information returned through the *lplpbBuffer* parameter.

Windows NT 4.0: This parameter must be zero.

Windows 2000 and later: This parameter should be zero, one, or two, corresponding to **RAS_CONNECTION_0**, **RAS_CONNECTION_1**, or **RAS_CONNECTION_2**.

lplpbBuffer
Upon successful execution, *lplpbBuffer* points to an array of structures that describe the enumerated connections. These structures are of type **RAS_CONNECTION_0**, **RAS_CONNECTION_1**, or **RAS_CONNECTION_2** depending on the value of the *dwLevel* parameter. Free this memory by calling **MprAdminBufferFree**.

dwPrefMaxLen
Preferred maximum length of returned data (in 8-bit bytes). If *dwPrefMaxLen* is -1, the buffer returned is large enough to hold all available information.

lpdwEntriesRead
Pointer to a **DWORD** variable. Upon successful return, this variable contains the total number of connections enumerated from the current resume position.

lpdwTotalEntries
Pointer to a **DWORD** variable. Upon successful return, this variable contains the total number of connections that could have been enumerated from the current resume position.

lpdwResumeHandle
Pointer to a **DWORD** variable. Upon successful return, this variable contains a resume handle that can be used to continue the enumeration. The *lpdwResumeHandle* parameter should be zero on the first call, and left unchanged on subsequent calls. If the return code is **ERROR_MORE_DATA,** another call may be made using this handle to retrieve more data. If the handle is NULL upon return, the enumeration cannot be continued. This handle is invalid for other types of error returns.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_INVALID_LEVEL	The value passed for *dwLevel* is not zero, one, or two. Levels one and two are supported only on Windows 2000 and later operating systems.
ERROR_INVALID_PARAMETER	At least one of the following parameters is NULL or does not point to valid memory: *lplpBuffer*, *lpdwEntriesRead*, or *lpdwTotalEntries*..
ERROR_MORE_DATA	Not all of the data was returned with this call. To obtain additional data, call the function again using the resume handle.
RPC_S_INVALID_BINDING	The handle passed in the *hRasServer* parameter is NULL or invalid..

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminConnectionEnum** rather than **MprAdminConnectionEnum**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminServerConnect**, **MprAdminBufferFree**, **RAS_CONNECTION_0**

MprAdminConnectionGetInfo

The **MprAdminConnectionGetInfo** function provides information on a specific connection.

```
DWORD MprAdminConnectionGetInfo(
  RAS_SERVER_HANDLE hRasServer,  // handle to server
  DWORD dwLevel,                 // level of info returned
  HANDLE hConnection,            // handle to connection
  LPBYTE *lplpbBuffer            // pointer to returned info
);
```

Parameters

hRasServer
Handle to the computer on which connection information is gathered. This computer should be running RRAS for Windows NT/Windows 2000. Obtain this handle by calling **MprAdminServerConnect**.

dwLevel
Specifies the format and content of the returned information. Acceptable values for *dwLevel* are zero or one. A value of zero returns a **RAS_CONNECTION_0** structure; a value of one returns a **RAS_CONNECTION_1** structure.

hConnection
Handle to the connection for which to obtain information. Obtain this handle by calling **MprAdminConnectionEnum**.

lplpbBuffer
Pointer to a pointer variable that points to a **RAS_CONNECTION_0** or **RAS_CONNECTION_1** structure upon successful execution. Free this memory by calling **MprAdminBufferFree**.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is ERROR_INVALID_PARAMETER.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminConnectionGetInfo** rather than **MprAdminConnectionGetInfo**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

! Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminServerConnect**, **MprAdminBufferFree**, **MprAdminConnectionEnum**, **RAS_CONNECTION_0**, **RAS_CONNECTION_1**

MprAdminPortClearStats

The **MprAdminPortClearStats** function resets the statistics for the specified port.

```
DWORD MprAdminPortClearStats(
  RAS_SERVER_HANDLE hRasServer,    // handle to the server
  HANDLE hPort                     // handle to the port
);
```

Parameters

hRasServer

Handle to the Remote Access Server on which to clear the statistics for the specified port. Obtain this handle by calling **MprAdminServerConnect**.

hPort

Handle to the port for which statistics are reset. Obtain this handle by calling **MprAdminPortEnum**.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is ERROR_INVALID_PARAMETER.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminPortClearStats** rather than **MprAdminPortClearStats**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminServerConnect**, **MprAdminPortEnum**

MprAdminPortDisconnect

The **MprAdminPortDisconnect** function disconnects a connection on a specific port.

```
DWORD MprAdminPortDisconnect(
  RAS_SERVER_HANDLE hRasServer,    // handle to the server
  HANDLE hPort                     // handle to the port
);
```

Parameters

hRasServer

Handle to the Remote Access Server on which to disconnect the port. Obtain this handle by calling **MprAdminServerConnect**.

hPort

Handle to the port to disconnect. Obtain this handle by calling **MprAdminPortEnum**.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is ERROR_INVALID_PARAMETER.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminPortDisconnect** rather than **MprAdminPortDisconnect**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminServerConnect**, **MprAdminPortEnum**

MprAdminPortEnum

The **MprAdminPortEnum** function enumerates all active ports in a specific connection, or all ports available for use or currently in use by RAS.

```
DWORD MprAdminPortEnum(
  RAS_SERVER_HANDLE hRasServer,  // handle to the server
  DWORD dwLevel,                 // must be zero
  HANDLE hConnection,            // handle to connection
  LPBYTE *lplpbBuffer,           // pointer to array of
                                 // port structs
  DWORD dwPrefMaxLen,            // maximum preferred length
                                 // of returned data
  LPDWORD lpdwEntriesRead,       // number of ports
                                 // enumerated
  LPDWORD lpdwTotalEntries,      // number of ports that
                                 // could've been enumerated
  LPDWORD lpdwResumeHandle       // handle with which to
                                 // resume enumeration
);
```

Parameters

hRasServer
Handle to the remote access server on which to enumerate ports. Obtain this handle by calling **MprAdminServerConnect**.

dwLevel
Specifies the level of information returned in the enumeration. This parameter must be zero.

hConnection
Handle to a connection within which the active ports are enumerated. If *hConnection* is **INVALID_HANDLE_VALUE**, all the ports in use or available for use by RRAS are enumerated. Obtain the *hConnection* handle by calling **MprAdminConnectionEnum**.

lplpbBuffer
Pointer to a pointer variable that will point to an array of **RAS_PORT_0** structures on successful return. Free this memory by calling **MprAdminBufferFree**.

dwPrefMaxLen
Preferred maximum length of returned data (in 8-bit bytes). If *dwPrefMaxLen* is -1, the buffer returned is large enough to hold all available information.

lpdwEntriesRead
Pointer to a **DWORD** variable. Upon successful return, this variable contains the total number of ports enumerated from the current resume position.

lpdwTotalEntries
Pointer to a **DWORD** variable. Upon successful return, this variable contains the total number of ports that could have been enumerated from the current resume position.

lpdwResumeHandle

Pointer to a **DWORD** variable. Upon successful execution, *lpdwResumeHandle* contains a handle that can be used to resume the enumeration. The *lpdwResumeHandle* parameter should be zero on the first call, and left unchanged on subsequent calls. If the return code is **ERROR_MORE_DATA,** the call may be reissued with the handle to retrieve more data. If the handle is NULL upon return, the enumeration cannot be continued. This handle is invalid for other types of error returns.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_ACCESS_DENIED	The calling application does not have sufficient privileges.
ERROR_DDM_NOT_RUNNING	The Demand Dial Manager (DDM) is not running, possibly because the Dynamic Interface Manager (DIM) is configured to run only on a LAN.
ERROR_INVALID_PARAMETER	At least one of the following parameters is NULL or does not point to valid memory: *lplpBuffer*, *lpdwEntriesRead*, or *lpdwTotalEntries*.
ERROR_MORE_DATA	Not all of the data was returned with this call. To obtain additional data, call the function again using the handle that was returned in the *lpdwResumeHandle* parameter.
ERROR_NOT_SUPPORTED	The *dwLevel* parameter is not zero.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminPortEnum** rather than **MprAdminPortEnum**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminBufferFree**, **MprAdminServerConnect**, **MprAdminConnectionEnum**

MprAdminPortGetInfo

The **MprAdminPortGetInfo** function gets information for a specific port.

```
DWORD MprAdminPortGetInfo(
  RAS_SERVER_HANDLE hRasServer,    // handle to the server
  DWORD dwLevel,                   // level of info returned
  HANDLE hPort,                    // handle to port
  LPBYTE *lplpbBuffer              // pointer returned data
);
```

Parameters

hRasServer
Handle to the Remote Access Server computer on which to collect port information. Obtain this handle by calling **MprAdminServerConnect**.

dwLevel
Specifies the format and content of the returned information. Acceptable values for *dwLevel* are zero or one. A value of zero will return a **RAS_PORT_0** structure; a value of one will return a **RAS_PORT_1** structure.

hPort
Handle to the port for which to collect information. Obtain this handle by calling **MprAdminPortEnum**.

lplpbBuffer
Pointer to a pointer variable that will point to a **RAS_PORT_0** or **RAS_PORT_1** structure on successful return. Free this memory by calling **MprAdminBufferFree**.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is ERROR_INVALID_PARAMETER.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminPortGetInfo** rather than **MprAdminPortGetInfo**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminServerConnect**, **MprAdminBufferFree**, **MprAdminPortEnum**

MprAdminPortReset

The **MprAdminPortReset** function resets the communication device attached to the specified port.

```
DWORD MprAdminPortReset(
  RAS_SERVER_HANDLE hRasServer,    // handle to the server
  HANDLE hPort                     // handle to the port
);
```

Parameters

hRasServer
Handle to the Remote Access Server on which to reset the specified port. Obtain this handle by calling **MprAdminServerConnect**.

hPort
Handle to the port to be reset. Obtain this handle by calling **MprAdminPortEnum**.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is ERROR_INVALID_PARAMETER.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminPortReset** rather than **MprAdminPortReset**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminServerConnect**, **MprAdminPortEnum**

RAS Admin DLL Functions

A RAS server administration DLL allows you to customize the following aspects of RAS:

- Access control for remote access clients
- Remote access client connection and disconnection event logging
- Logging and control of IP address allocation to remote access clients.

A RAS Admin DLL must implement and export all of the following functions:

MprAdminAcceptNewLink
MprAdminConnectionHangupNotification
MprAdminConnectionHangupNotification2
MprAdminGetIpAddressForUser
MprAdminLinkHangupNotification
MprAdminReleaseIpAddress

In addition, the RAS Admin DLL must implement and export either

MprAdminAcceptNewConnection, and
MprAdminConnectionHangupNotification

or

MprAdminAcceptNewConnection2, and
MprAdminConnectionHangupNotification2

If not all of the required functions are implemented, the remote access service will fail to start.

RAS serializes calls into the administration DLL. A call into one of the DLL's functions for a given RAS client will never be preempted by a call to that function for a different RAS client; the initial call is guaranteed to complete before RAS calls the function for the other client. Furthermore, serialization extends to certain groups of functions. The IP address functions are serialized as a group; a call into either **MprAdminGetIpAddressForUser** or **MprAdminReleaseIpAddress** will block calls into both until the initial call completes. Together, the new connection/link and connection/link-hang-up notification functions are also serialized as a group.

Do not call any of the RAS Administration Functions or Ras User Administration Functions from inside a callout function. Calls to these functions will not return when made from within a callout function.

MprAdminAcceptNewConnection

Remote Access Service calls the **MprAdminAcceptNewConnection** function each time a new user dials in and successfully completes RAS authentication. **MprAdminAcceptNewConnection** determines whether the user is allowed to connect.

```
BOOL MprAdminAcceptNewConnection(
  RAS_CONNECTION_0 *pRasConnection0,
            // struct that describes connection
  RAS_CONNECTION_1 *pRasConnection1
            // struct that describes connection
);
```

Parameters

pRasConnection0
Pointer to a **RAS_CONNECTION_0** structure describing this connection.

pRasConnection1
Pointer to a **RAS_CONNECTION_1** structure describing this connection.

Return Values

If **MprAdminAcceptNewConnection** accepts the connection, the return value should be TRUE.

If **MprAdminAcceptNewConnection** rejects the connection, the return value should be FALSE.

Remarks

If **MprAdminAcceptNewConnection** does not accept the new connection, RAS will not call the **MprAdminConnectionHangupNotification** function.

Do not call any of the RAS Administration Functions or Ras User Administration Functions from inside **MprAdminAcceptNewConnection**. Calls to these functions will not return when made from within a callout function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminAcceptNewConnection2**, **MprAdminConnectionHangupNotification**, **MprAdminConnectionHangupNotification2**, **RAS_CONNECTION_0**, **RAS_CONNECTION_1**

MprAdminAcceptNewConnection2

Remote Access Service calls the **MprAdminAcceptNewConnection2** function each time a new user dials in and successfully completes RAS authentication. **MprAdminAcceptNewConnection2** determines whether the user is allowed to connect.

```
BOOL MprAdminAcceptNewConnection2(
  RAS_CONNECTION_0 *pRasConnection0,
  RAS_CONNECTION_1 *pRasConnection1,
  RAS_CONNECTION_2 *pRasConnection2
);
```

Parameters

pRasConnection0
Pointer to a **RAS_CONNECTION_0** structure describing this connection.

pRasConnection1
Pointer to a **RAS_CONNECTION_1** structure describing this connection.

pRasConnection2
Pointer to a **RAS_CONNECTION_2** structure describing this connection.

Return Values

If **MprAdminAcceptNewConnection2** accepts the connection, the return value should be TRUE.

If **MprAdminAcceptNewConnection2** rejects the connection, the return value should be FALSE.

Remarks

If MprAdminAcceptNewConnection2 does not accept the new connection, RAS will not call the **MprAdminConnectionHangupNotification2** function.

Do not call any of the RAS Administration Functions or Ras User Administration Functions from inside **MprAdminAcceptNewConnection2**. Calls to these functions will not return when made from within a callout function.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminConnectionHangupNotification2**, **RAS_CONNECTION_0**, **RAS_CONNECTION_1**, **RAS_CONNECTION_2**

MprAdminAcceptNewLink

RAS calls the **MprAdminAcceptNewLink** function each time a link is created for a particular connection. RAS calls this function once immediately after **MprAdminAcceptNewConnection** returns, and an additional time for every new link that is to be used with the connection.

```
BOOL MprAdminAcceptNewLink(
  RAS_PORT_0 *pRasPort0,   // struct that describes the port
  RAS_PORT_1 *pRasPort1    // struct that describes the port
);
```

Parameters

pRasPort0
Pointer to a **RAS_PORT_0** structure that describes the port being used by the link.

pRasPort1
Pointer to a **RAS_PORT_1** structure that describes the port being used by the link.

Return Values

If RAS should accept the new link, the return value should be TRUE.

If RAS should not accept the new link, the return value should be FALSE.

Remarks

If RAS does not accept the new link, RAS will not call the **MprAdminLinkHangupNotification** function.

Do not call any of the RAS Administration Functions or RAS User Administration Functions from inside **MprAdminAcceptNewLink**. Calls to these functions will not return when made from within a callout function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminAcceptNewConnection**, **MprAdminConnectionHangupNotification**, **MprAdminLinkHangupNotification**, **RAS_PORT_0**, **RAS_PORT_1**

MprAdminConnectionHangupNotification

Remote Access Service calls the **MprAdminConnectionHangupNotification** function after the last link for the specified connection has been dismantled.

```
VOID MprAdminConnectionHangupNotification(
  RAS_CONNECTION_0 *pRasConnection0,
       // struct that describes connection
  RAS_CONNECTION_1 *pRasConnection1
       // struct that describes connection
);
```

Parameters

pRasConnection0
: Pointer to a **RAS_CONNECTION_0** structure describing this connection.

pRasConnection1
: Pointer to a **RAS_CONNECTION_1** structure describing this connection.

Return Values

This function does not have a return value.

Remarks

Do not call any of the RAS Administration Functions or Ras User Administration Functions from inside **MprAdminConnectionHangupNotification**. Calls to these functions will not return when made from within a callout function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminAcceptNewConnection**, **MprAdminAcceptNewLink**, **MprAdminConnectionHangupNotification2**, **RAS_CONNECTION_0**, **RAS_CONNECTION_1**

MprAdminConnectionHangupNotification2

Remote Access Service calls the **MprAdminConnectionHangupNotification2** function after the last link for the specified connection has been dismantled.

```
VOID MprAdminConnectionHangupNotification2(
  RAS_CONNECTION_0 *pRasConnection0,
  RAS_CONNECTION_1 *pRasConnection1,
  RAS_CONNECTION_2 *pRasConnection2
  );
```

Parameters

pRasConnection0
Pointer to a **RAS_CONNECTION_0** structure describing this connection.

pRasConnection1
Pointer to a **RAS_CONNECTION_1** structure describing this connection.

pRasConnection2
Pointer to a **RAS_CONNECTION_2** structure describing this connection.

Return Values

This function does not have a return value.

Remarks

Do not call any of the RAS Administration Functions or Ras User Administration Functions from inside **MprAdminConnectionHangupNotification2**. Calls to these functions will not return when made from within a callout function.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminAcceptNewConnection2**, **MprAdminAcceptNewLink**, **RAS_CONNECTION_0**, **RAS_CONNECTION_1**, **RAS_CONNECTION_2**

MprAdminGetIpAddressForUser

RAS calls **MprAdminGetIpAddressForUser** once for each user that requires an IP address. RAS calls the function with the IP address that RAS selects for the user. The third-party DLL that implements this function may change this address to one of its own choosing.

```
DWORD MprAdminGetIpAddressForUser(
  WCHAR *lpwszUserName,  // pointer to username
  WCHAR *lpwszPortName,  // pointer to port name
  DWORD *lpdwIpAddress,  // pointer to IP address
  BOOL *bNotifyRelease   // notify DLL when user disconnects
);
```

Parameters

lpwszUserName
Pointer to a Unicode string containing the name of the user requiring an IP address.

lpwszPortName
Pointer to a Unicode string containing the name of the port on which the user is attempting to connect.

lpdwIpAddress
Pointer to a **DWORD** variable. When RAS calls the function, this variable contains either the IP address RAS intends to allocate for the user or zero. If the variable contains an IP address, the DLL can either accept the address or substitute a different one. If the variable contains a zero, the DLL must allocate an IP address for the user. If this variable is zero, and the DLL does not allocate an IP address, the user will not be able to connect.

bNotifyRelease
Pointer to a **BOOL** variable. If the DLL sets this variable to TRUE, RAS will call **MprAdminReleaseIpAddress** when the user disconnects. Otherwise, RAS will not notify the DLL when this IP address is released.

Return Values

If function succeeds, the return value should be NO_ERROR.

If the function returns anything other than NO_ERROR, RAS will terminate the connection.

Remarks

Do not call any of the RAS Administration Functions or Ras User Administration Functions from inside **MprAdminGetIpAddressForUser**. Calls to these functions will not return when made from within a callout function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminReleaseIpAddress**

MprAdminLinkHangupNotification

RAS calls the **MprAdminLinkHangupNotification** function whenever a link for a particular connection is dismantled.

```
VOID MprAdminLinkHangupNotification(
  RAS_PORT_0 *pRasPort0,    // struct that describes the port
  RAS_PORT_1 *pRasPort1     // struct that describes the port
);
```

Parameters

pRasPort0
Pointer to a **RAS_PORT_0** structure that describes the port being used by the link.

pRasPort1
Pointer to a **RAS_PORT_1** structure that describes the port being used by the link.

Return Values

This function does not have a return value.

Remarks

Do not call any of the RAS Administration Functions or Ras User Administration Functions from inside **MprAdminLinkHangupNotification**. Calls to these functions will not return when made from within a callout function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminConnectionHangupNotification**, **MprAdminConnectionHangupNotification2**, **MprAdminAcceptNewConnection**, **MprAdminAcceptNewConnection2**, **MprAdminAcceptNewLink**, **RAS_PORT_0**, **RAS_PORT_1**

MprAdminReleaseIpAddress

The **MprAdminReleaseIpAddress** function is called when a user disconnects and the user's IP address is about to be released.

```
VOID MprAdminReleaseIpAddress(
  WCHAR *lpwszUserName,    // pointer to username
  WCHAR *lpwszPortName,    // pointer to port name
  DWORD *lpdwIpAddress     // pointer to IP address
);
```

Parameters

lpwszUserName
: Pointer to a Unicode string containing the name of the user requiring an IP address.

lpwszPortName
: Pointer to a Unicode string containing the name of the port on which the user is attempting to connect.

lpdwIpAddress
: Pointer to a **DWORD** variable. This variable contains the IP address to be released.

Return Values

This function does not have a return value.

Remarks

Do not call any of the RAS Administration Functions or Ras User Administration Functions from inside **MprAdminReleaseIpAddress**. Calls to these functions will not return when made from within a callout function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminConnectionHangupNotification**, **MprAdminGetIpAddressForUser**

RAS User Administration Functions

Use the following functions to manage dial-up users:

MprAdminGetPDCServer
MprAdminSendUserMessage
MprAdminUserGetInfo
MprAdminUserSetInfo

To obtain a list of current users on a particular domain, use the **NetQueryDisplayInformation** function. The prototype for this function is in the lmaccess.h header file.

MprAdminGetPDCServer

The **MprAdminGetPDCServer** function retrieves the name of the server with the master User Accounts Subsystem (UAS) from either a domain name or a server name. Either the domain name parameter or the server name parameter may be NULL, but not both.

```
DWORD MprAdminGetPDCServer(
  const WCHAR * lpwsDomainName,    // pointer to domain name
  const WCHAR * lpwsServerName,    // pointer to server name
  LPWSTR lpwsPDCName               // pointer to buffer to
                                   // receive name of server
                                   // with UAS
);
```

Parameters

lpwsDomainName
Pointer to a null-terminated Unicode string that contains the name of the domain to which the RAS server belongs. This parameter can be NULL if you are running your RAS administration application on a Windows NT/Windows 2000 Server that is not participating in a domain. If this parameter is NULL, the *lpwsServerName* parameter must not be NULL.

lpwsServerName
Pointer to a null-terminated Unicode string that contains the name of the Windows NT/Windows 2000 RAS server. Specify the name with leading "\\" characters, in the form: *servername*. This parameter can be NULL if the *lpwsDomain* parameter is not NULL.

lpwsPDCName

Pointer to a buffer that receives a null-terminated Unicode string containing the name of a domain controller that has the user account database. The buffer should be big enough to hold the server name (UNCLEN +1). The function prefixes the returned server name with leading "\\" characters, in the form: *servername*.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails the return value is one of the following values.

Value	Meaning
ERROR_NO_SUCH_DOMAIN	The domain specified is not valid.
NERR_InvalidComputer	The *lpwsDomainName* is NULL, and *lpwsServerName* parameter is not valid.

Remarks

The **MprAdminGetPDCServer** function can obtain the name of the server with the user accounts database given the name of the RAS server, or the name of the domain in which the RAS server resides. To get the server name, call the **GetComputerName** function

If the server name specified by *lpszServer* is part of a domain, The server returned by **MprAdminGetPDCServer** will be either the primary domain controller or a backup domain controller.

If the server name specified by *lpszServer* is a stand-alone Windows NT/Windows 2000 Server (that is, the server or workstation does not participate in a Windows NT/Windows 2000 domain), then the server name itself is returned in the *lpszUserAccountServer* buffer.

You can then use the name of the user account server in a call to the **NetQueryDisplayInformation** function to enumerate the users in the user account database. You can also use the server name in calls to the **MprAdminUserGetInfo** and **MprAdminUserSetInfo** functions to get and set RAS privileges for a specified user account.

! Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **GetComputerName**, **MprAdminUserGetInfo**, **MprAdminUserSetInfo**, **NetQueryDisplayInformation**

MprAdminSendUserMessage

The **MprAdminSendUserMessage** function sends a message to the user connected on the specified connection.

```
DWORD MprAdminSendUserMessage(
  HANDLE hConnection,    // handle to connection
  LPWSTR lpwszMessage    // pointer to message
);
```

Parameters

hConnection
Handle to the connection on which the user is connected. Use **MprAdminConnectionEnum** to obtain this handle.

lpwszMessage
Pointer to a Unicode string containing the message to the user.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is one of the following error codes.

Value	Meaning
ERROR_ACCESS_DENIED	The caller does not have sufficient privileges.
ERROR_DDM_NOT_RUNNING	The Demand Dial Manager (DDM) is not running, possibly because the Dynamic Interface Manager (DIM) is configured to run only on a LAN.
ERROR_INTERFACE_NOT_CONNECTED	The *hConnection* parameter is not valid.
ERROR_INVALID_PARAMETER	The *lpwszMessage* parameter is NULL.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminConnectionEnum**

MprAdminUserGetInfo

The **MprAdminUserGetInfo** function retrieves all RAS information for a particular user.

```
DWORD MprAdminUserGetInfo(
  const WCHAR *lpwsServerName,     // name of PDC or BDC
                                   // with UAS
  const WCHAR *lpwsUserName,       // name of user
  DWORD dwLevel,                   // must be zero
  LPBYTE lpbBuffer                 // RAS_USER_0 structure
);
```

Parameters

lpwsServerName

Pointer to a Unicode string containing the name of the server computer with the master User Accounts Subsystem (UAS). If the remote access server is part of a domain, the computer with the UAS will be either the primary domain controller or the backup domain controller. If the remote access server is not part of a domain, then the server itself will store the UAS. In either case, call the **MprAdminGetPDCServer** function to obtain the value for this parameter.

If the server itself stores the UAS, this parameter may be NULL.

lpwsUserName

Pointer to a Unicode string containing the name of the user for which to get RAS information.

dwLevel

This parameter must be zero.

Windows 2000 and later: This parameter may be zero or one.

lpbBuffer

Pointer to a **RAS_USER_0** structure. The caller must allocate (and free) the memory for this structure. Upon successful return, this structure contains the RAS data for the specified user.

Windows 2000 and later: If the *dwLevel* parameter specifies one, *lpbBuffer* should point to a **RAS_USER_1** structure.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails the return value is one of the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The caller does not have sufficient privileges.
ERROR_INVALID_LEVEL	The value of *dwLevel* is invalid.
ERROR_INVALID_PARAMETER	*lpbBuffer* is NULL
ERROR_NO_SUCH_USER	The user specified by *lpwsUserName* does not exist on the server specified by *lpwsServerName*.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminUserGetInfo** rather than **MprAdminUserGetInfo**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminGetPDCServer**, **MprAdminUserSetInfo**, **RAS_USER_0**

MprAdminUserSetInfo

The **MprAdminUserSetInfo** function sets RAS information for the specified user.

```
DWORD MprAdminUserSetInfo(
  const WCHAR *lpwsServerName,    // name of PDC or BDC
                                  // with UAS
  const WCHAR *lpwsUserName,      // name of user
  DWORD dwLevel,                  // must be zero
  const LPBYTE lpbBuffer          // RAS_USER_0 structure
);
```

Parameters

lpwsServerName

Pointer to a Unicode string containing the name of the server computer with the master User Accounts Subsystem (UAS). If the remote access server is part of a domain, the computer with the UAS will be either the primary domain controller or the backup domain controller. If the remote access server is not part of a domain, then the server itself will store the UAS. In either case, call the **MprAdminGetPDCServer** function to obtain the value for this parameter.

If the server itself stores the UAS, this parameter may be NULL.

lpwsUserName

Pointer to a Unicode string containing the name of the user for which to set RAS information.

dwLevel

This parameter must be zero.

Windows 2000 and later: This parameter may be zero or one.

lpbBuffer

Pointer to a **RAS_USER_0** structure that specifies the new RAS information for the user.

Windows 2000 and later: If the *dwLevel* parameter specifies one, *lpbBuffer* should point to a **RAS_USER_1** structure.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value is one of the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The caller does not have sufficient privileges.
ERROR _INVALID_LEVEL	The value of *dwLevel* is invalid.
ERROR_NOT_ENOUGH_MEMORY	There are insufficient resources to complete the operation.
ERROR_NO_SUCH_USER	The user specified by *lpwsUserName* does not exist on the server specified by *lpwsServerName*.

Remarks

This function is available on Windows NT 4.0 if the RRAS redistributable is installed. However, the version of Mprapi.dll that ships with the RRAS redistributable exports the function as **RasAdminUserSetInfo** rather than **MprAdminUserSetInfo**. Therefore, when using the RRAS redistributable, use **LoadLibrary** and **GetProcAddress** to access this function.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.
Library: Use Mprapi.lib.

See Also

Remote Access Service Administration Reference, RAS Administration Functions, **MprAdminGetPDCServer**, **MprAdminUserGetInfo**, **RAS_USER_0**

RAS Administration Structures

The RAS Administration Functions use the following structures:

PPP_ATCP_INFO
PPP_CCP_INFO
PPP_INFO
PPP_INFO_2
PPP_IPCP_INFO
PPP_IPCP_INFO2
PPP_IPXCP_INFO
PPP_LCP_INFO
PPP_NBFCP_INFO
RAS_CONNECTION_0
RAS_CONNECTION_1
RAS_CONNECTION_2
RAS_PORT_0
RAS_PORT_1
RAS_USER_0
RAS_USER_1

PPP_ATCP_INFO

The **PPP_ATCP_INFO** structure contains the result of a PPP AppleTalk projection operation.

```
typedef struct _PPP_ATCP_INFO {
  DWORD    dwError;
  WCHAR    wszAddress[ ATADDRESSLEN + 1 ];
} PPP_ATCP_INFO;
```

Members

dwError

Specifies the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation.

wszAddress

Specifies a Unicode string that holds the client's AppleTalk address on the RAS connection.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **PPP_INFO**

PPP_CCP_INFO

The **PPP_CCP_INFO** structure contains information that describes the results of a Compression Control Protocol (CCP) negotiation.

```
typedef struct _PPP_CCP_INFO {
  OUT DWORD          dwError;
  OUT DWORD          dwCompressionAlgorithm;
  OUT DWORD          dwOptions;
  OUT DWORD          dwRemoteCompressionAlgorithm;
  OUT DWORD          dwRemoteOptions;
} PPP_CCP_INFO;
```

Members

dwError

Specifies an error if the negotiation is unsuccessful.

dwCompressionAlgorithm

Specifies the compression algorithm that the local computer is using. The following table shows the possible values for this member.

Value	Meaning
RASCCPCA_MPPC	Microsoft Point-to-Point Compression (MPPC) Protocol
RASCCPCA_STAC	STAC option 4

dwOptions

Specifies the compression options on the local computer. The following options are supported:

Option	Meaning
PPP_CCP_COMPRESSION	Compression without encryption.
PPP_CCP_HISTORYLESS	Microsoft Point to Point Encryption (MPPE) in stateless mode. The session key is changed after every packet. This mode improves performance on high latency networks, or networks that experience significant packet loss.
PPP_CCP_ENCRYPTION40BITOLD	MPPE using 40-bit keys.
PPP_CCP_ENCRYPTION40BIT	MPPE using 40-bit keys.
PPP_CCP_ENCRYPTION56BIT	MPPE using 56-bit keys.
PPP_CCP_ENCRYPTION128BIT	MPPE using 128-bit keys.

dwRemoteCompressionAlgorithm

Specifies the compression algorithm that the remote computer is using. The following table shows the possible values for this member.

Value	Meaning
RASCCPCA_MPPC	Microsoft Point-to-Point Compression (MPPC) Protocol
RASCCPCA_STAC	STAC option 4

dwRemoteOptions

Specifies the compression options on the remote computer. The following options are supported.

Option	Meaning
PPP_CCP_COMPRESSION	Compression without encryption.
PPP_CCP_HISTORYLESS	Microsoft Point to Point Encryption (MPPE) in stateless mode. The session key is changed after every packet. This mode improves performance on high latency networks, or networks that experience significant packet loss.
PPP_CCP_ENCRYPTION40BITOLD	MPPE using 40-bit keys.
PPP_CCP_ENCRYPTION40BIT	MPPE using 40-bit keys.
PPP_CCP_ENCRYPTION56BIT	MPPE using 56-bit keys.
PPP_CCP_ENCRYPTION128BIT	MPPE using 128-bit keys.

! Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.

See Also

PPP_LCP_INFO

PPP_INFO

The **PPP_INFO** structure is used to report the results of the various PPP projection operations for a connection.

```
typedef struct _PPP_INFO {
  PPP_NBFCP_INFO    nbf;
  PPP_IPCP_INFO     ip;
  PPP_IPXCP_INFO    ipx;
  PPP_ATCP_INFO     at;
} PPP_INFO;
```

Members

nbf
Specifies a **PPP_NBFCP_INFO** structure.

ip
Specifies a **PPP_IPCP_INFO** structure.

ipx
Specifies a **PPP_IPXCP_INFO** structure.

at
Specifies a **PPP_ATCP_INFO** structure.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **PPP_ATCP_INFO**, **PPP_IPCP_INFO**, **PPP_IPXCP_INFO**, **PPP_NBFCP_INFO**, **RAS_CONNECTION_1**

PPP_INFO_2

The **PPP_INFO_2** structure is used to report the results of the various PPP projection operations for a connection.

```
typedef struct _PPP_INFO_2 {
  PPP_NBFCP_INFO  nbf;
  PPP_IPCP_INFO   ip;
  PPP_IPXCP_INFO  ipx;
  PPP_ATCP_INFO   at;
  PPP_CCP_INFO    ccp;
  PPP_LCP_INFO    lcp;
} PPP_INFO_2;
```

Members

nbf
Specifies a **PPP_NBFCP_INFO** structure.

ip
Specifies a **PPP_IPCP_INFO** structure.

ipx
Specifies a **PPP_IPXCP_INFO** structure.

at
Specifies a **PPP_ATCP_INFO** structure.

ccp
Specifies a **PPP_CCP_INFO** structure.

lcp
Specifies a **PPP_LCP_INFO** structure.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **PPP_ATCP_INFO**, **PPP_IPCP_INFO**, **PPP_IPXCP_INFO**, **PPP_NBFCP_INFO**, **PPP_CCP_INFO**, **PPP_LCP_INFO**, **RAS_CONNECTION_2**

PPP_IPCP_INFO

The **PPP_IPCP_INFO** structure contains the result of a PPP Internet Protocol (IP) negotiation.

```
typedef struct _PPP_IPCP_INFO {
  DWORD    dwError;
  WCHAR    wszAddress[ IPADDRESSLEN + 1 ];
  WCHAR    wszRemoteAddress[ IPADDRESSLEN + 1 ];
} PPP_IPCP_INFO;
```

Members

dwError

Specifies the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation.

wszAddress

Specifies a Unicode string that holds the local computer's IP address for the connection. This string has the form a.b.c.d; for example, "11.101.237.71".

The **PPP_IPCP_INFO** structures provides address information from the perspective of the server. For example, if a remote access client is connecting to a RAS server, this member holds the IP address of the server.

wszRemoteAddress

Specifies a Unicode string that holds the IP address of the remote computer. This string has the form "a.b.c.d". If the address is not available, this member is an empty string, "".

The **PPP_IPCP_INFO** structures provides address information from the perspective of the server. For example, if a remote access client is connecting to a RAS server, this member holds the IP address of the client.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **PPP_INFO**, **PPP_IPCP_INFO2**

PPP_IPCP_INFO2

The **PPP_IPCP_INFO2** structure contains the result of a PPP Internet Protocol (IP) negotiation.

```
typedef struct _PPP_IPCP_INFO2 {
  DWORD    dwError;
  WCHAR    wszAddress[ IPADDRESSLEN + 1 ];
  WCHAR    wszRemoteAddress[ IPADDRESSLEN + 1 ];
  DWORD    dwOptions;
  DWORD    dwRemoteOptons;
} PPP_IPCP_INFO2;
```

Members

dwError

Specifies the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation.

wszAddress

Specifies a Unicode string that holds the local computer's IP address for the connection.

The **PPP_IPCP_INFO2** structures provides address information from the perspective of the server. For example, if a remote access client is connecting to a RAS server, this member holds the IP address of the server.

wszRemoteAddress

Specifies a Unicode string that holds the IP address of the remote computer. If the address is not available, this member specifies an empty string, "".

The **PPP_IPCP_INFO2** structures provides address information from the perspective of the server. For example, if a remote access client is connecting to a RAS server, this member holds the IP address of the client.

dwOptions

Specifies IPCP options for the local computer. Currently, the only option is PPP_IPCP_VJ. This option indicates that IP datagrams sent by the local computer are compressed using Van Jacobson compression.

dwRemoteOptons

Specifies IPCP options for the remote peer. Currently, the only option is PPP_IPCP_VJ. This option indicates that IP datagrams sent by the remote peer (that is, received by the local computer) are compressed using Van Jacobson compression.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **PPP_INFO**, **PPP_IPCP_INFO**

PPP_IPXCP_INFO

The **PPP_IPXCP_INFO** structure contains the result of a PPP Internetwork Packet Exchange (IPX) projection operation.

```
typedef struct _PPP_IPXCP_INFO {
  DWORD    dwError;
  WCHAR    wszAddress[ IPXADDRESSLEN + 1 ];
} PPP_IPXCP_INFO;
```

Members

dwError

Specifies the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation.

wszAddress

Specifies a Unicode string that holds the client's IPX address on the RAS connection.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **PPP_INFO**

PPP_LCP_INFO

The **PPP_LCP_INFO** structure contains information that describes the results of an PPP Link Control Protocol (LCP) negotiation.

```
typedef struct _PPP_LCP_INFO {
  OUT DWORD dwError;
  OUT DWORD dwAuthenticationProtocol;
  OUT DWORD dwAuthenticationData;
  OUT DWORD dwRemoteAuthenticationProtocol;
  OUT DWORD dwRemoteAuthenticationData;
  OUT DWORD dwTerminateReason;
  OUT DWORD dwRemoteTerminateReason;
  OUT DWORD dwOptions;
  OUT DWORD dwRemoteOptions;
  OUT DWORD dwEapTypeId;
  OUT DWORD dwRemoteEapTypeId;
} PPP_LCP_INFO;
```

Members

dwError

Specifies the error that occurred if the negotiation was unsuccessful.

dwAuthenticationProtocol

Specifies the authentication protocol used to authenticate the local computer. This member can be one of the following values.

Value	Meaning
PPP_LCP_PAP	Password Authentication Protocol
PPP_LCP_SPAP	Shiva Password Authentication Protocol
PPP_LCP_CHAP	Challenge Handshake Authentication Protocol
PPP_LCP_EAP	Extensible Authentication Protocol

dwAuthenticationData

Specifies additional information about the authentication protocol specified by the **dwAuthenticationProtocol** member. This member can be one of the following values.

Value	Meaning
PPP_LCP_CHAP_MD5	MD5 CHAP
PPP_LCP_CHAP_MS	Microsoft CHAP
PPP_LCP_CHAP_MSV2	Microsoft CHAP version 2

dwRemoteAuthenticationProtocol

Specifies the authentication protocol used to authenticate the remote computer. See the **dwAuthenticationProtocol** member for a list of possible values.

dwRemoteAuthenticationData

Specifies additional information about the authentication protocol specified by **dwRemoteAuthenticationProtocol**. See the **dwAuthenticationData** member for a list of possible values.

dwTerminateReason

This member always has a value of zero.

dwRemoteTerminateReason

This member always has a value of zero.

dwOptions

Specifies information about LCP options in use by the local computer. This member is a combination of the following flags.

Flag	Meaning
PPP_LCP_MULTILINK_FRAMING	The connection is using multilink
RASLCPO_PFC	Protocol Field Compression
RASLCPO_ACFC	Address and Control Field Compression

(continued)

(continued)

Flag	Meaning
RASLCPO_SSHF	
RASLCPO_DES_56	DES 56-bit encryption
RASLCPO_3_DES	Triple DES Encryption

dwRemoteOptions
Specifies information about LCP options in use by the remote computer. This member is a combination of the following flags.

Flag	Meaning
PPP_LCP_MULTILINK_FRAMING	The connection is using multilink.
RASLCPO_PFC	Protocol Field Compression (see *RFC 1172*)
RASLCPO_ACFC	Address and Control Field Compression (see *RFC 1172*)
RASLCPO_SSHF	Short Sequence Number Header Format (see *RFC 1990*)
RASLCPO_DES_56	DES 56-bit encryption
RASLCPO_3_DES	Triple DES Encryption

dwEapTypeId
Specifies the type identifier of the Extensible Authentication Protocol (EAP) used to authenticate the local computer. The value of this member is valid only if **dwAuthenticationProtocol** is PPP_LCP_EAP.

dwRemoteEapTypeId
Specifies the type identifier of the Extensible Authentication Protocol (EAP) used to authenticate the remote computer. The value of this member is valid only if **dwRemoteAuthenticationProtocol** is PPP_LCP_EAP.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.

See Also

PPP_CCP_INFO

PPP_NBFCP_INFO

The **PPP_NBFCP_INFO** structure contains the result of a PPP NetBEUI Framer (NBF) projection operation.

```
typedef struct _PPP_NBFCP_INFO {
  DWORD    dwError;
  WCHAR    wszWksta[ NETBIOS_NAME_LEN + 1 ];
} PPP_NBFCP_INFO;
```

Members

dwError

Specifies the result of the PPP control protocol negotiation. A value of zero indicates success. A nonzero value indicates failure, and is the actual fatal error that occurred during the control protocol negotiation.

wszWksta

Specifies a Unicode string that is the local workstation's computer name. This unique computer name is the closest NetBIOS equivalent to a client's NetBEUI address on a remote access connection.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **PPP_INFO**

RAS_CONNECTION_0

The **RAS_CONNECTION_0** structure contains general information regarding a specific connection, such as user name or domain. For more detailed information about a specific connection, such as bytes sent or received, see **RAS_CONNECTION_1**.

```
typedef struct _RAS_CONNECTION_0 {
  HANDLE                   hConnection;
  HANDLE                   hInterface;
  DWORD                    dwConnectDuration;  // In seconds
  ROUTER_INTERFACE_TYPE    dwInterfaceType;
  DWORD                    dwConnectionFlags;
  WCHAR                    wszInterfaceName[ MAX_INTERFACE_NAME_LEN + 1 ];
  WCHAR                    wszUserName[ UNLEN + 1 ];
  WCHAR                    wszLogonDomain[ DNLEN + 1 ];
  WCHAR                    wszRemoteComputer[ NETBIOS_NAME_LEN + 1 ];

} RAS_CONNECTION_0, *PRAS_CONNECTION_0;
```

Members

hConnection
Handle to the connection.

hInterface
Handle to the interface.

dwConnectDuration
Specifies the duration of the current connection, in seconds.

dwInterfaceType
Specifies the interface type of the current connection.

dwConnectionFlags
Specifies one of a set of flags that describe this connection. This member can contain the following flags.

Flag	Meaning
RAS_FLAGS_PPP_CONNECTION	The connection is using Point-to-Point Protocol (PPP).
RAS_FLAGS_MESSENGER_PRESENT	The messenger service is active on the client, and that messages can be sent to the client using **MprAdminSendUserMessage**.
RAS_FLAGS_RAS_CONNECTION	The connection is a NetBIOS connection from a Windows 3.11 or Windows for Workgroups client.
RAS_FLAGS_ARAP_CONNECTION	The connection is using AppleTalk Remote Access Protocol (ARAP).

wszInterfaceName
Specifies a unicode string that contains the name of the interface for this connection.

wszUserName
Specifies a unicode string that contains the name of the user that is logged on to the connection.

wszLogonDomain
Specifies a unicode string that contains the domain which the connected user is logged onto.

wszRemoteComputer
Specifies a unicode string that contains the name of the remote computer.

! Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **RAS_CONNECTION_1**, **RAS_CONNECTION_2**

RAS_CONNECTION_1

The **RAS_CONNECTION_1** structure contains detailed information regarding a specific connection, such as error counts and bytes received. For more general information about a specific connection, such as user name or domain, see ***RAS_CONNECTION_0***.

```
typedef struct RAS_CONNECTION_1 {
  HANDLE      hConnection;
  HANDLE      hInterface;
  PPP_INFO    PppInfo;
  DWORD       dwBytesXmited;
  DWORD       dwBytesRcved;
  DWORD       dwFramesXmited;
  DWORD       dwFramesRcved;
  DWORD       dwCrcErr;
  DWORD       dwTimeoutErr;
  DWORD       dwAlignmentErr;
  DWORD       dwHardwareOverrunErr;
  DWORD       dwFramingErr;
  DWORD       dwBufferOverrunErr;
  DWORD       dwCompressionRatioIn;
  DWORD       dwCompressionRatioOut;
} RAS_CONNECTION_1; *PRAS_CONNECTION_1;
```

Members

hConnection
Handle to the connection.

hInterface
Handle to the interface.

pppInfo
Specifies a **PPP_INFO** structure.

dwBytesXmited
Specifies the bytes transmitted on the current connection.

dwBytesRcved
Specifies the bytes received on the current connection.

dwFramesXmited
Specifies the frames transmitted on the current connection.

dwFramesRcved
Specifies the frames received on the current connection.

dwCrcErr
Specifies the CRC (Cyclic Redundancy Check) errors on the current connection.

dwTimeoutErr
Specifies the time-out errors on the current connection.

dwAlignmentErr
Specifies the alignment errors on the current connection.

dwHardwareOverrunErr
Specifies the number of hardware overrun errors on the current connection.

dwFramingErr
Specifies the number of framing errors for the current connection.

dwBufferOverrunErr
Specifies the number of buffer overrun errors.

dwCompressionRatioIn
Specifies a percentage that indicates the degree to which data received on this connection is compressed. The ratio is the size of the compressed data divided by the size of the same data in an uncompressed state.

dwCompressionRatioOut
Specifies a percentage that indicates the degree to which data transmitted on this connection is compressed. The ratio is the size of the compressed data divided by the size of the same data in an uncompressed state.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **RAS_CONNECTION_0**, **RAS_CONNECTION_2**, **PPP_INFO**

RAS_CONNECTION_2

The **RAS_CONNECTION_2** structure contains information for a connection, including the GUID that identifies the connection.

```
typedef struct _RAS_CONNECTION_2 {
  HANDLE      hConnection;
                // handle to the connection
  WCHAR       wszUserName[ UNLEN + 1 ];
                // name of the user on this connection
```

```
  ROUTER_INTERFACE_TYPE   dwInterfaceType;
                // interface type for the connection
  GUID          guid;
                // guid that identifies the connection
  PPP_INFO_2  PppInfo2;
} RAS_CONNECTION_2, * PRAS_CONNECTION_2;
```

Members

hConnection
Handle to the connection.

wszUserName[UNLEN + 1]
Specifies a unicode string that contains the name of the user on this connection.

dwInterfaceType
Specifies the type of interface.

guid
Specifies a GUID (Globally Unique IDentifier) that identifies the connection. For incoming connection, this GUID is valid only as long as the connection is active.

PppInfo2
Specifies a **PPP_INFO_2** structure that contains information about the PPP negotiation for this connection.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **MprAdminConnectionEnum**, **RAS_CONNECTION_0**, **RAS_CONNECTION_1**

RAS_PORT_0

The **RAS_PORT_0** structure contains general information regarding a specific RAS port, such as port condition and port name. For more detailed information about a specific port, such as line speed or errors, see ***RAS_PORT_1***.

```
typedef struct _RAS_PORT_0 {
  HANDLE                hPort;
  HANDLE                hConnection;
  RAS_PORT_CONDITION    dwPortCondition;
  DWORD                 dwTotalNumberOfCalls;
  DWORD                 dwConnectDuration;    // In seconds
  WCHAR                 wszPortName[ MAX_PORT_NAME + 1 ];
```

(continued)

(continued)

```
    WCHAR               wszMediaName[ MAX_MEDIA_NAME + 1 ];
    WCHAR               wszDeviceName[ MAX_DEVICE_NAME + 1 ];
    WCHAR               wszDeviceType[ MAX_DEVICETYPE_NAME + 1 ];

} RAS_PORT_0, *PRAS_PORT_0;
```

Members

hPort
Handle to the port.

hConnection
Handle to the connection.

dwPortCondition
RAS_PORT_CONDITION structure.

dwTotalNumberOfCalls
Specifies the cumulative number of calls this port has serviced.

dwConnectDuration
Specifies the duration of the current connection, in seconds.

wszPortName
Specifies the port name.

wszMediaName
Specifies the media name.

wszDeviceName
Specifies the device name.

wszDeviceType
Specifies the device type.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **RAS_PORT_1**, **RAS_PORT_CONDITION**

RAS_PORT_1

The **RAS_PORT_1** structure contains detailed information regarding a specific RAS port, such as line speed or errors. For more general information about a port, such as port condition or port name, see ***RAS_PORT_0***.

```
typedef struct _RAS_PORT_1{
   HANDLE                      hPort;
   HANDLE                      hConnection;
   RAS_HARDWARE_CONDITION      dwHardwareCondition;
   DWORD                       dwLineSpeed;   // in bits/second
   DWORD                       dwBytesXmited;
   DWORD                       dwBytesRcved;
   DWORD                       dwFramesXmited;
   DWORD                       dwFramesRcved;
   DWORD                       dwCrcErr;
   DWORD                       dwTimeoutErr;
   DWORD                       dwAlignmentErr;
   DWORD                       dwHardwareOverrunErr;
   DWORD                       dwFramingErr;
   DWORD                       dwBufferOverrunErr;
   DWORD                       dwCompressionRatioIn;
   DWORD                       dwCompressionRatioOut;
} RAS_PORT_1, *PRAS_PORT_1;
```

Members

hPort
Handle to the port.

hConnection
Handle to the connection.

dwHardwareCondition
Specifies a **RAS_HARDWARE_CONDITION** structure.

dwLineSpeed
Specifies the line speed of the port, represented in bits per second.

dwBytesXmited
Specifies the bytes transmitted on the port.

dwBytesRcved
Specifies the bytes received on the port.

dwFramesXmited
Specifies the frames transmitted on the port.

dwFramesRcved
Specifies the frames received on the port.

dwCrcErr
Specifies the CRC errors on the port.

dwTimeoutErr
Specifies the time-out errors on the port.

dwAlignmentErr
Specifies the alignment errors on the port.

dwHardwareOverrunErr
Specifies the hardware overrun errors on the port.

dwFramingErr
Specifies the framing errors on the port.

dwBufferOverrunErr
Specifies the buffer overrun errors on the port.

dwCompressionRatioIn
Specifies a percentage that indicates the degree to which data received on this connection is compressed. The ratio is the size of the compressed data divided by the size of the same data in an uncompressed state.

dwCompressionRatioOut
Specifies a percentage indicating the degree to which data transmitted on this connection is compressed. The ratio is the size of the compressed data divided by the size of the same data in an uncompressed state.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **RAS_PORT_0**, **RAS_HARDWARE_CONDITION**

RAS_USER_0

The **RAS_USER_0** structure contains information for a particular Remote Access Service user.

```
typedef struct _RAS_USER_0 {
  BYTE     bfPrivilege;     // RASPRIV flags
  WCHAR    wszPhoneNumber[ MAX_PHONE_NUMBER_LEN + 1];
} RAS_USER_0, *PRAS_USER_0;
```

Members

bfPrivilege
Specifies the types of remote access privilege available to the RAS user.

The following remote access privilege constants are defined in Mprapi.h.

Value	Meaning
RASPRIV_DialinPrivilege	The user has permission to dial-in to the RAS server.
RASPRIV_NoCallback	The RAS server will not call back the user to establish a connection.
RASPRIV_AdminSetCallback	When the user calls, the RAS server hangs up and calls a preset call-back phone number stored in the user account database. The **wszPhoneNumber** member of the **RAS_USER_0** structure contains the user's call-back phone number.
RASPRIV_CallerSetCallback	When the user calls, the RAS server provides the option of specifying a call-back phone number. The user can also choose to connect immediately without a call back. The **wszPhoneNumber** member contains a default number that the user can override.

Use the following constant as a mask to isolate the call-back privilege. (This constant is also defined in Mprapi.h.)

RASPRIV_CallbackType

wszPhoneNumber

Pointer to a Unicode string containing the phone number at which the RAS user should be called back.

Requirements

Windows NT/2000: Requires Windows NT 4.0 or later.
Header: Declared in Rassapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **MprAdminUserGetInfo**, **MprAdminUserSetInfo**, **RAS_USER_1**

RAS_USER_1

The **RAS_USER_1** structure contains information for a particular Remote Access Service user. The **RAS_USER_1** structure is similar to the **RAS_USER_0** structure, except that **RAS_USER_1** supports an additional member, **bfPrivilege2**.

```
typedef struct _RAS_USER_1 {
  OUT BYTE        bfPrivilege;
  OUT WCHAR       wszPhoneNumber[ MAX_PHONE_NUMBER_LEN + 1];
  OUT BYTE        bfPrivilege2;
} RAS_USER_1, *PRAS_USER_1;
```

Members

bfPrivilege

Specifies the types of remote access privilege available to the RAS user.

The following remote access privilege constants are defined in Mprapi.h.

Value	Meaning
RASPRIV_DialinPrivilege	The user has permission to dial-in to the RAS server.
RASPRIV_NoCallback	The RAS server will not call back the user to establish a connection.
RASPRIV_AdminSetCallback	When the user calls, the RAS server hangs up and calls a preset call-back phone number stored in the user account database. The **wszPhoneNumber** member of the **RAS_USER_0** structure contains the user's call-back phone number.
RASPRIV_CallerSetCallback	When the user calls, the RAS server provides the option of specifying a call-back phone number. The user can also choose to connect immediately without a call back. The **wszPhoneNumber** member contains a default number that the user can override.

Use the following constant as a mask to isolate the call back privilege. (This constant is also defined in Mprapi.h.)

RASPRIV_CallbackType

wszPhoneNumber

Pointer to a Unicode string containing the phone number at which the RAS user should be called back.

bfPrivilege2

Specifies flags specifying additional remote access privileges that are available to the RAS user.

The following remote access privilege constants are defined in Mprapi.h.

Value	Meaning
RASPRIV2_DialinPolicy	Remote access policies determine whether the user is allowed dial-in access.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Structures, **MprAdminUserGetInfo**, **MprAdminUserSetInfo**, **RAS_USER_0**

RAS Administration Enumerated Types

The RAS Administration Functions use the following enumerated types:

RAS_HARDWARE_CONDITION

RAS_PORT_CONDITION

RAS_HARDWARE_CONDITION

The **RAS_HARDWARE_CONDITION** enumeration type specifies hardware status information about a given RAS port.

```
typedef enum _RAS_HARDWARE_CONDITION {
  RAS_HARDWARE_OPERATIONAL,
  RAS_HARDWARE_FAILURE

} RAS_HARDWARE_CONDITION;
```

Values

RAS_HARDWARE_OPERATIONAL
The port is operational.

RAS_HARDWARE_FAILURE
The port is not operational, due to a hardware failure.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Enumerated Types

RAS_PORT_CONDITION

The **RAS_PORT_CONDITION** enumerated type specifies information regarding the connection condition of a given RAS port.

```
typedef enum _RAS_PORT_CONDITION
{
  RAS_PORT_NON_OPERATIONAL,
  RAS_PORT_DISCONNECTED,
  RAS_PORT_CALLING_BACK,
  RAS_PORT_LISTENING,
  RAS_PORT_AUTHENTICATING,
  RAS_PORT_AUTHENTICATED,
  RAS_PORT_INITIALIZING

} RAS_PORT_CONDITION;
```

Values

RAS_PORT_NON_OPERATIONAL
The port is not operational.

RAS_PORT_DISCONNECTED
The port is disconnected.

RAS_PORT_CALLING_BACK
The port is in the process of a call back.

RAS_PORT_LISTENING
The port is listening for incoming calls.

RAS_PORT_AUTHENTICATING
The port is authenticating a user.

RAS_PORT_AUTHENTICATED
The port has authenticated a user.

RAS_PORT_INITIALIZING
The port is initializing.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Mprapi.h.

See Also

Remote Access Service Administration Reference, RAS Administration Enumerated Types

CHAPTER 13

Extensible Authentication Protocol (EAP)

EAP Overview

Microsoft® Windows® 2000 supports the Extensible Authentication Protocol (EAP). EAP allows third-party authentication modules to interact with the implementation of the Point-to-Point Protocol (PPP) included in Windows 2000 Remote Access Service (RAS).

EAP is an extension to PPP, providing a standard support mechanism for authentication schemes such as token cards, Kerberos, Public Key, and S/Key. EAP has been made available in response to increasing demand to augment RAS authentication with third-party security devices.

EAP is fully supported on both the Windows 2000 Dial-Up Server and the Dial-Up Networking Client. EAP is a critical technology component for secure Virtual Private Networks (VPN), protecting them against "brute force" or "dictionary" attacks and password guessing.

EAP improves on previous authentication protocols such as Password Authentication Protocol (PAP) and Challenge Handshake Authentication Protocol (CHAP). Windows 2000 supports these earlier authentication protocols as well.

EAP and Internet Authentication Service

The Extensible Authentication Protocol (EAP) is supported on RAS servers running Microsoft® Windows® 2000. It is also supported on Windows 2000 Servers running Internet Authentication Service (IAS). IAS provides remote authentication services using Remote Access Dial-In User Service (RADIUS). The following documentation is applicable to implementing an EAP on a RAS server or on an IAS server. If you are implementing EAP on IAS, simply treat references to RAS as though they refer to IAS.

EAP Installation

Vendors implement EAPs, also known as authentication protocols, in Dynamic-Link Libraries (DLLs). A DLL for the authentication protocol must reside on both the client and server computers. For simplicity, the client and server DLLs may be identical; however, this is not a requirement. Also, note that the same DLL may support more than one authentication protocol.

The vendor should provide setup software to install and remove the DLL. The setup software should also create the appropriate keys and values for the authentication protocol in the system registry. The installation of each EAP DLL should create the following registry key.

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Rasman\PPP\EAP\<eaptypeid>

In the preceding path, **<eaptypeid>** is the identifier of the authentication protocol. The vendor must obtain this identifier from the Internet Assigned Numbers Authority (IANA).

The setup software should remove this key when uninstalling the DLL. The system removes this key if the user uninstalls RAS.

For a description of the supported values for this key, see *Authentication Protocol Registry Values*.

Authentication Protocol Registry Values

The setup software for the EAP DLL may create the following registry values below **<eaptypeid>**. These registry values are defined in the Raseapif.h header file. The RAS_EAP_VALUENAME_PATH and RAS_EAP_VALUENAME_FRIENDLY_NAME values are required. The setup software may create other keys and values as well. These could be used by the authentication protocol itself. For an example of registry configuration, see *Registry Values Example*.

RAS_EAP_VALUENAME_PATH

RAS_EAP_VALUENAME_FRIENDLY_NAME

RAS_EAP_VALUENAME_CONFIGUI

RAS_EAP_VALUENAME_DEFAULT_DATA

RAS_EAP_VALUENAME_REQUIRE_CONFIGUI

RAS_EAP_VALUENAME_CONFIG_CLSID

RAS_EAP_VALUENAME_IDENTITY

RAS_EAP_VALUENAME_INTERACTIVEUI

RAS_EAP_VALUENAME_INVOKE_NAMEDLG

RAS_EAP_VALUENAME_INVOKE_PWDDLG

RAS_EAP_VALUENAME_ENCRYPTION

RAS_EAP_VALUENAME_STANDALONE_SUPPORTED

RAS_EAP_VALUENAME_PATH

Constant Value	Path
Type	REG_EXPAND_SZ
Description	Specifies the path to the EAP DLL.

RAS_EAP_VALUENAME_FRIENDLY_NAME

Constant Value FriendlyName

Type REG_SZ

Description Specifies a friendly name for the authentication protocol. This name will appear in the Dial-Up Networking user interface.

RAS_EAP_VALUENAME_CONFIGUI

Constant Value ConfigUIPath

Type REG_EXPAND_SZ

Description Specifies the path to the DLL that implements the configuration user interface.

RAS_EAP_VALUENAME_DEFAULT_DATA

Constant Value ConfigData

Type REG_BINARY

Description Specifies default configuration data for the authentication protocol.

RAS_EAP_VALUENAME_REQUIRE_CONFIGUI

Constant Value RequireConfigUI

Type REG_DWORD

Description Specifies whether the user must provide configuration data in the Dial-Up Networking user interface. If this value is 1, the user will not be allowed to exit the Dial-Up Networking UI without providing configuration data. The default value is 0.

RAS_EAP_VALUENAME_CONFIG_CLSID

Constant Value ConfigCLSID

Type REG_SZ

Description Specifies the class identifier of the configuration UI on the server.

RAS_EAP_VALUENAME_IDENTITY

Constant Value IdentityPath

Type REG_EXPAND_SZ

Description Specifies the path to the DLL that implements functions to obtain the user's identity.

RAS_EAP_VALUENAME_INTERACTIVEUI

Constant Value	InteractiveUIPath
Type	REG_EXPAND_SZ
Description	Specifies the path to the DLL that implements the interactive user interface.

RAS_EAP_VALUENAME_INVOKE_NAMEDLG

Constant Value	InvokeUsernameDialog
Type	REG_DWORD
Description	Specifies whether the RAS Connection Manager should display the standard Windows NT/2000 user name dialog (value of 1) or invoke **RasEapGetIdentity** (value of 0). The default value is 1.

RAS_EAP_VALUENAME_INVOKE_PWDDLG

Constant Value	InvokePasswordDialog
Type	REG_DWORD
Description	Specifies whether the RAS Connection Manager should display the standard Windows NT/2000 password dialog. If this value exists and is 0, RAS will not display the password dialog. The default value is 1.

RAS_EAP_VALUENAME_ENCRYPTION

Constant Value	MPPEEncryptionSupported
Type	REG_DWORD
Description	If this value is 1, the authentication protocol can generate keys for the Microsoft Point-to-Point Encryption (MPPE) style of encryption. Possible values are 0 or 1. The default value is 0.

RAS_EAP_VALUENAME_STANDALONE_SUPPORTED

Constant Value	StandaloneSupported
Type	REG_DWORD
Description	Specifies whether this authentication protocol is supported on stand-alone Windows 2000 servers. A value of 0 indicates that the EAP is not supported. The default value is 1.

Registry Values Example

The following example shows possible data for some of the authentication protocol registry values.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Rasman\PPP\EAP\40
  Path               (REG_EXPAND_SZ)  %SystemRoot%\system32\sample.dll
  FriendlyName       (REG_SZ)         Sample EAP Protocol
  ConfigUIPath       (REG_EXPAND_SZ)  %SystemRoot%\system32\sample.dll
  IdentityPath       (REG_EXPAND_SZ)  %SystemRoot%\system32\sample.dll
  InteractiveUIPath  (REG_EXPAND_SZ)  %SystemRoot%\system32\sample.dll
  RequireConfigUI    (REG_DWORD)      1
  ConfigCLSID        (REG_SZ)   {0000031A-0000-0000-C000-000000000046}
  StandaloneSupported  (REG_DWORD)    1
```

User Authentication

The authentication protocol may authenticate the user itself. EAP-TLS is an example of such a protocol. Alternatively, the authentication protocol may rely on a separate authentication provider to authenticate the user. Two authentication providers are built into Microsoft® Windows® 2000: Windows 2000 domain authentication (accessed via Directory Services) and RADIUS (Remote Access Dial In User Service).

In the case where RADIUS is the authentication provider, the EAP DLL is installed on the RADIUS server rather than on the RAS server. The RAS server passes EAP packets directly from the client to the authentication protocol on the RADIUS server. The RAS server does not process any of the information in the EAP packets.

EAP Implementation Details

Microsoft® Windows® 2000 RAS interacts with EAP implementations through the use of function calls that must be exported by the third-party EAP DLL. This interaction is detailed in the following topics:

- RAS Connection Manager Initialization
- Authentication Protocol Initialization
- RAS and Authentication Protocol Interaction
- Completion of the Authentication Session

RAS Connection Manager Initialization

After initialization, the Remote Access Service (RAS) Connection Manager queries the registry for installed authentication protocols. RAS calls the exported function **RasEapGetInfo** one time for each authentication protocol. The **RasEapGetInfo** function receives a single parameter of type **PPP_EAP_INFO**. RAS uses the **dwEapTypeId** member of this structure to specify the authentication protocol (note that a single DLL may support more than one protocol). If **RasEapGetInfo** returns any value other than NO_ERROR, RAS assumes that the authentication protocol is unavailable.

On return from **RasEapGetInfo** the **PPP_EAP_INFO** structure contains pointers to the functions **RasEapInitialize**, **RasEapBegin**, **RasEapMakeMessage**, and **RasEapEnd** in the EAP DLL. RAS uses these functions to interoperate with the authentication protocol. RAS immediately calls **RasEapInitialize** for each authentication protocol, to initialize it. When RAS shuts down it calls **RasEapInitialize** again, this time with a value of FALSE, indicating that the authentication protocol should shut itself down.

Authentication Protocol Initialization

To create a phone-book entry for a particular connection, the user selects an authentication protocol to use for that connection. The selected authentication protocol may require configuration. If so, the Dial-Up Networking user interface (UI) displays a configuration UI by calling the **RasEapInvokeConfigUI** function. The Dial-Up Networking UI stores the configuration information returned by **RasEapInvokeConfigUI** in the phone-book entry. The setup program for the authentication protocol may also store default *configuration* information in the registry. For more information, see *EAP Installation*.

The configuration information stored in the phone-book entry should be generic to all users on the client computer. Information specific to a particular user or users should not be stored in the phone-book entry. The authentication protocol should obtain user-specific information via the identity function interface or interactive user-interface. The authentication protocol can store this information in the registry by passing it to RAS in the *pEapOutput* parameter of **RasEapMakeMessage**.

The configuration information should not be specific to the current machine; it should be portable from machine to machine.

When the client attempts to establish the connection, RAS obtains identity information for the user. If the RAS_EAP_VALUENAME_INVOKE_NAMEDLG value is present in the registry for this authentication protocol, and this value is set to zero, RAS calls **RasEapGetIdentity**. This function typically displays a user interface that allows the identity information to be of a type specific to the authentication protocol; for example, a certificate or numeric ID. If RAS_EAP_VALUENAME_INVOKE_NAMEDLG is not present, or is set to one, RAS displays the standard Windows NT/Windows 2000 user-name dialog.

Once RAS has obtained the identity information for the user, RAS calls the authentication protocol's implementation of **RasEapBegin**. This call allows the protocol to allocate and initialize a work buffer that RAS passes on subsequent calls to **RasEapMakeMessage** and **RasEapEnd**. In **RasEapBegin**, RAS also passes a **PPP_EAP_INPUT** structure that contains pointers to the configuration information for the connection, and the identity information for the user. RAS always passes in a value for the **pszIdentity** member of **PPP_EAP_INPUT**. However, the **pszPassword** member of **PPP_EAP_INPUT** may be NULL.

Within the **PPP_EAP_INPUT** structure, the **fAuthenticator** member indicates whether the authentication protocol is being invoked to be authenticated (on the client) or as the authenticator (on the server).

On the server, the **bInitialID** member of **PPP_EAP_INPUT** specifies the identifier that the server should use for the first EAP packet. The server should increment this identifier for subsequent packets.

Also on the server, the **pUserAttributes** pointer in **PPP_EAP_INPUT** points to an array of attributes of the **RAS_AUTH_ATTRIBUTE_TYPE** type. These are attributes for the user that were obtained from the client.

If the **RasEapBegin** call returns any value other than NO_ERROR, the session is disconnected. The returned error is logged (on the server), or displayed to the user (on the client).

RAS and Authentication Protocol Interaction During Authentication

The **RasEapMakeMessage** function controls the majority of the interaction between the authentication protocol and the RAS Connection Manager. **RasEapMakeMessage** processes incoming EAP packets and creates EAP packets for transmission to the remote peer. It also processes events such as time outs and authentication completion.

If a message is received from the remote peer, RAS calls **RasEapMakeMessage**, passing a pointer to the received message in the *pReceivePacket* parameter.

If RAS calls **RasEapMakeMessage** with *pReceivePacket* set to NULL, RAS is either initiating the dialog by using the authentication protocol, or requesting that the protocol resend the last packet. The authentication protocol should determine which action RAS is taking based on its state and from the message context.

On return from **RasEapMakeMessage**, the value of the **Action** member of the **PPP_EAP_OUTPUT** structure indicates what action, if any, RAS should take. The **Action** member takes values from the PPP_EAP_ACTION enumerated type.

If **Action** is EAPACTION_Send, EAPACTION_SendAndDone, EAPACTION_SendWithTimeout, or EAPACTION_SendWithTimeoutInteractive, the RAS Connection Manager transmits the packet that is pointed to by the *pSendPacket* parameter to the remote peer.

If **Action** is EAPACTION_SendWithTimeout, or EAPACTION_SendWithTimeoutInteractive, the authentication protocol should set the **dwIdExpected** member of the **PPP_EAP_OUTPUT** structure to the identifier of the next packet that is expected from the remote peer. Regardless of whether the next packet received from the peer matches this value, RAS passes the packet to the authentication protocol in a subsequent call to **RasEapMakeMessage**. The authentication protocol may silently discard the packet by simply returning ERROR_PPP_INVALID_PACKET. If a packet that has the expected identifier is not received within the configured time-out period, RAS calls **RasEapMakeMessage**. The call is made with the *pReceivePacket* parameter set to NULL, to indicate that the previous packet must be sent again.

The EAPACTION_SendWithTimeout value allows for a time out, after which time the RAS Connection Manager Service disconnects the session.

The EAPACTION_SendWithTimeoutInteractive value provides for an infinite amount of time out to occur. The authenticator should use this value when expecting user input on the client. This time out allows the user an unspecified amount of time to complete the required input.

If the **Action** member is EAPACTION_Done or EAPACTION_SendAndDone, RAS examines the **dwAuthResultCode** member of **PPP_EAP_OUTPUT**. If **dwAuthResultCode** is NO_ERROR, the authentication succeeded. If **dwAuthResultCode** is a value other than NO_ERROR, the authentication failed. The error code returned for the failure case should come from Raserror.h, Mprerror.h, or Winerror.h. Possible return codes include, but are not limited to, the following:

ERROR_NO_DIALIN_PERMISSION

ERROR_PASSWD_EXPIRED

ERROR_ACCT_DISABLED

ERROR_RESTRICTED_LOGON_HOURS

ERROR_AUTH_INTERNAL

In the case where **Action** is EAPACTION_Done or EAPACTION_SendAndDone, the **pUserAttributes** member should point to attributes that override attributes of the same type that were passed to the server in the call to **RasEapBegin**.

The authentication protocol can request that RAS invoke the current authentication provider by returning EAPACTION_Authenticate in the **Action** member in **PPP_EAP_OUTPUT**. In this case, the **pUserAttributes** pointer in **PPP_EAP_OUTPUT** should point only to attributes that were generated by the authentication protocol on the server. It need not include any of the attributes that were passed to the server in the call to **RasEapBegin**. When RAS responds to the **EAPACTION_Authenticate** action, **pUserAttributes** (in **PPP_EAP_INPUT**), will point to all attributes generated during authentication. These attributes will also be returned to the authentication protocol on the client.

If the authentication protocol authenticates the user without relying on an authentication provider, there is no need for the protocol to ever set **Action** to EAPACTION_Authenticate. An example of this case is EAP-TLS.

The EAP success packet is not acknowledged. Therefore, it may be lost and not resent by the server. If the RAS Connection Manager on the client receives a Network Control Protocol (NCP) packet, RAS is programmed to proceed as though the authentication was successful, but the EAP success packet was lost. This is because the server has moved on to the NCP phase of PPP. Accordingly, RAS calls **RasEapMakeMessage** with the **fSuccessPacketReceived** member of the **PPP_EAP_INPUT** structure set to TRUE.

During the course of the authentication session, the authentication protocol may need to interact directly with the user on the client. The authentication protocol vendor can provide an interactive user interface for this purpose. The authentication protocol can request that RAS display the interactive UI by setting the **fInvokeInteractiveUI**, **pUIContextData**, and **dwSizeOfUIContextData** members in the **PPP_EAP_OUTPUT** structure. For more information on using an interactive UI, see *Interactive User Interface*.

The authentication protocol should display a user interface only through the mechanism described under Interactive User Interface. If the authentication protocol itself displays the user interface, the PPP thread blocks until the user interface is dismissed.

If during the authentication process, **RasEapMakeMessage** returns any value other than NO_ERROR or ERROR_PPP_INVALID_PACKET, the session is disconnected and the error is logged (on the server) or displayed to the user (on the client).

Completion of the Authentication Session

After the authentication session is completed, the RAS Connection Manager calls the **RasEapEnd** function to allow the authentication protocol to deallocate its work buffer. This action is taken regardless of whether authentication was successful. Calling the **RasEapEnd** function guarantees that no further calls are made to the authentication protocol using that particular user or context without first calling **RasEapBegin**.

Configuration User Interface

Configuration user interfaces (UI) for authentication protocols are implemented differently depending on whether the UI configures the authentication protocol on the client, or on the server. The following topics describe the process used to implement a configuration UI for the client and for the server:

- Server-Side Configuration User Interface
- Client-Side Configuration User Interface

Server-Side Configuration User Interface

Implement a configuration UI for the server by implementing the COM interface, **IEAPProviderConfig**. This COM interface derives from **IUnknown** and adds three methods: **IEAPProviderConfig::Initialize**, **IEAPProviderConfig::ServerInvokeConfigUI**, and **IEAPProviderConfig::Uninitialize**.

The UI should support remote administration. In other words, although the UI will configure the authentication protocol on the server, the UI itself may be running on a different computer. To support remote administration, separate the UI code from the code that actually performs the configuration. (The configuration code resides on the server on which the authentication protocol runs.)

Microsoft recommends using the Active Template Library (ATL) to implement **IEAPProviderConfig**. See the sample server-side configuration UI in the SDK samples directory for more details. The CLSID for the configuration UI object should be placed in the registry with a value name of RAS_EAP_VALUENAME_CONFIG_CLSID. (For more information, see ***Authentication Protocol Registry Values***.)

When the user clicks the Configure button for an authentication protocol (in the Properties dialog box for Routing and RAS), the system checks if a RAS_EAP_VALUENAME_CONFIG_CLSID for this authentication protocol exists in the registry. If so, COM instantiates the configuration UI object. If the system is unable to find RAS_EAP_VALUENAME_CONFIG_CLSID in the registry, and the system has access to Directory Services (DS) (Windows 2000 only), the system attempts to instantiate the object from the Class Store.

In the case where the user is connected to multiple machines simultaneously, multiple configuration UI objects are instantiated.

Client-Side Configuration User Interface

The vendor that implements the authentication protocol may also provide a configuration User Interface (UI) for the protocol. The configuration UI may be implemented in the same DLL as the authentication protocol, or in a separate DLL. Also, the DLL that implements the configuration UI may support more than one authentication protocol. The path to the DLL for the configuration user interface is stored in the RAS_EAP_VALUENAME_CONFIGUI registry value, under the key for the authentication protocol. For more information about creating this registry value, see *EAP Installation*.

The DLL for the configuration user interface should export entry points for the following functions:

RasEapInvokeConfigUI

RasEapFreeMemory

When the user creates a phone-book entry for a particular RAS server in the Dial-Up-Networking UI, the user is able to select the authentication protocol that RAS should use with that entry. If the authentication protocol is configurable, the Dial-Up-Networking UI calls **RasEapInvokeConfigUI** to invoke the configuration UI. The Dial-Up-Networking UI stores the configuration information returned by **RasEapInvokeConfigUI** in the phone-book entry

The configuration information stored in the phone-book entry should be generic to all users on the client computer. Information specific to a particular user or users should not be stored in the phone-book entry. The authentication protocol should obtain user-specific information by using the identity functions or interactive user-interface. The authentication protocol can store this information in the registry by passing it to RAS in the *pEapOutput* parameter of **RasEapMakeMessage**.

The configuration information should also not be specific to the current machine; it should be portable from machine to machine.

When RAS calls the **RasEapBegin** function for the authentication protocol, it passes a **PPP_EAP_INPUT** structure that contains a pointer to the configuration information. After **RasEapBegin** returns, RAS calls **RasEapFreeMemory** to free the memory occupied by the configuration information. Therefore, the authentication protocol should copy the configuration information into a private memory buffer during the call to **RasEapBegin**.

The vendor may add a value under the registry key for the authentication protocol that specifies default configuration information for the protocol. The vendor may also add a value that specifies whether the user is required to enter configuration information when they create a phone-book entry. For more information, see *Authentication Protocol Registry Values*.

Obtaining Identity Information

The vendor that implements the authentication protocol may also provide a function interface that obtains initial identifying information for the user requesting authentication. The vendor should implement the following functions:

RasEapGetIdentity

RasEapFreeMemory

These functions may be implemented in the same DLL as the authentication protocol, or in a separate DLL. Also, the DLL that implements the identity functions may support more than one authentication protocol. The path to the DLL for these functions is stored in the RAS_EAP_VALUENAME_IDENTITY registry value, under the key for the authentication protocol. For more information about creating this registry value, see *EAP Installation*.

The **RasEapGetIdentity** function typically displays a User Interface (UI) to obtain identity information for the user. However, if the *dwFlags* parameter contains the RAS_EAP_FLAG_NON_INTERACTIVE flag, **RasEapGetIdentity** should not display a UI.

If **RasEapGetIdentity** does display a UI, the UI must support **WM_COMMAND** messages where the value of **LOWORD**(*wParam*) is equal to IDCANCEL.

The RAS Connection Manager calls **RasEapGetIdentity** if the RAS_EAP_VALUENAME_INVOKE_NAMEDLG value that is in the registry for this EAP is set to zero. If RAS_EAP_VALUENAME_INVOKE_NAMEDLG is not present, or is present and is set to one, RAS displays the standard Windows NT/Windows 2000 user name dialog box.

In addition to RAS_EAP_VALUENAME_INVOKE_NAMEDLG, the EAP vendor may create a related value in the registry: RAS_EAP_VALUENAME_INVOKE_PWDDLG. If this value is present and is set to zero, RAS will not display the standard Windows NT/Windows 2000 password dialog. This value can be useful to implement a biometric method such as a fingerprint scan to authenticate the user. If both the RAS_EAP_VALUENAME_INVOKE_NAMEDLG and RAS_EAP_VALUENAME_INVOKE_PWDDLG values are zero, an identity UI can be used to obtain both the identity and biometric information. However, if only RAS_EAP_VALUENAME_INVOKE_PWDDLG is zero, RAS will not call **RasEapGetIdentity**. In this case, use the interactive user interface to obtain the biometric information.

For more information on these registry values, see *Authentication Protocol Registry Values*.

The information obtained by **RasEapGetIdentity** is passed to the authentication protocol during the call to **RasEapBegin**. The information is pointed to by the **pszIdentity** and **pUserData** members of the **PPP_EAP_INPUT** structure. To save this information in the registry on the client computer, the authentication protocol should return the information in the *pEapOutput* parameter of **RasEapMakeMessage**.

After the call to **RasEapBegin**, RAS calls **RasEapFreeMemory** to free the memory occupied by this data. Therefore, the authentication protocol should copy the information into a private memory buffer during the call to **RasEapBegin**.

Interactive User Interface

The vendor that implements the authentication protocol may also provide an interactive User Interface (UI) for the protocol. The interactive UI allows the authentication protocol to obtain additional information from the user as needed during the course of the authentication session.

The interactive UI can be implemented in the same DLL as the authentication protocol, or in a separate DLL. Also, the DLL that implements the interactive UI can support more than one authentication protocol. The path to the DLL for the interactive UI is stored in the RAS_EAP_VALUENAME_INTERACTIVEUI registry value, under the key for the authentication protocol. For more information about creating this registry value, see *EAP Installation*.

The DLL for the interactive UI should export entry points for the following functions:

RasEapInvokeInteractiveUI

RasEapFreeMemory

The interactive user interface must support **WM_COMMAND** messages where **LOWORD**(*wParam*) equals IDCANCEL.

To display the interactive UI, the authentication protocol should set the **fInvokeInteractiveUI** member of the **PPP_EAP_OUTPUT** structure to TRUE. The authentication protocol may optionally set the **pUIContextData** and **dwSizeOfUIContextData** members as well. RAS uses the values of these members to pass context data to the interactive UI. The authentication protocol returns this **PPP_EAP_OUTPUT** structure as a parameter in the **RasEapMakeMessage** function.

RAS invokes the interactive UI by calling **RasEapInvokeInteractiveUI**. RAS passes the authentication protocol a pointer to the data that was returned by the interactive UI in the subsequent call to **RasEapMakeMessage**. The pointer is passed as a member of a **PPP_EAP_INPUT** structure. After **RasEapMakeMessage** returns, RAS calls **RasEapFreeMemory** to free the memory occupied by the information. Therefore, the authentication protocol should copy the information into a private memory buffer during the call to **RasEapMakeMessage**.

Multilink and Callback Connections

For the first link in a multilink connection, RAS sets the RAS_EAP_FLAG_FIRST_LINK flag in the **fFlags** member of the **PPP_EAP_INPUT** structure. The authentication protocol can use the presence of this flag to determine whether to present a user interface specifically for the first link of a multilink connection.

If the connection is configured so that the server calls back the client computer, the RAS_EAP_FLAG_FIRST_LINK flag will not be set on the callback.

If the authentication protocol sets the **fSaveConnectionData** member of **PPP_EAP_OUTPUT** to TRUE, subsequent links in the multilink connection will receive the new connection-specific data. In the case of user-specific data, however, the authentication protocol continues to get the original user-specific data even if it sets the **fSaveUserData** member of **PPP_EAP_OUTPUT** to TRUE.

The authentication protocol may use an interactive user interface to collect data for a particular link of a multilink connection. In this case, RAS makes the resulting data available to the authentication protocol during subsequent links. This data is never saved to persistent storage, however.

EAP Reference

This section describes the reference elements that are used to implement the Extensible Authentication Protocol (EAP). Among these reference elements are functions that you can use to program authentication protocols, authentication providers, and accounting providers. This section also includes the structures and enumerated types that these functions use.

EAP Functions

Implement the following functions for authentication protocols and authentication providers:

RasEapBegin
RasEapEnd
RasEapFreeMemory
RasEapGetIdentity
RasEapGetInfo
RasEapInvokeConfigUI
RasEapInvokeInteractiveUI
RasEapMakeMessage

RasEapBegin

The Connection Manager calls the **RasEapBegin** function to initiate an authentication session.

```
DWORD ( *RasEapBegin )(
  VOID * * ppWorkBuffer,          // buffer used in
                                  // subsequent
                                  // calls to protocol
  PPP_EAP_INPUT * pPppEapInput    // initialization
                                  // information
);
```

Parameters

ppWorkBuffer
Pointer to a pointer that, on successful return, points to a work buffer. This buffer is opaque to RAS; the contents of the buffer are used only by the authentication protocol. The Connection Manager passes a pointer to this buffer to the authentication protocol in subsequent calls to **RasEapMakeMessage**.

pPppEapInput
Pointer to a **PPP_EAP_INPUT** structure that contains initialization information for the authentication session.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h.

Remarks

The **RasEapBegin** function is not part of the RRAS API; it is implemented in the EAP DLL. When the Connection Manager calls the **RasEapGetInfo** function, it receives a **PPP_EAP_INFO** structure for the authentication protocol. This structure contains a pointer to the **RasEapBegin** function.

The memory for the work buffer (pointed to by **ppWorkBuffer*) is allocated by the authentication protocol. The authentication protocol should free this memory in its implementation of **RasEapEnd**.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Functions, **RasEapEnd**, **RasEapGetInfo**, **RasEapMakeMessage**, **PPP_EAP_INFO**, **PPP_EAP_INPUT**

RasEapEnd

The Connection Manager calls the **RasEapEnd** function to end an authentication session. RAS will call **RasEapEnd** regardless of whether the session completed successfully.

```
DWORD ( *RasEapEnd ) (
  VOID * pWorkBuffer    // work buffer to free
);
```

Parameters

pWorkBuffer
Pointer to the work buffer to free.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h. If **RasEapEnd** returns an error code, RAS terminates the authentication session.

Remarks

The **RasEapEnd** function is not part of the RRAS API; it is implemented in the EAP DLL. When the Connection Manager calls the **RasEapGetInfo** function, it receives a **PPP_EAP_INFO** structure for the authentication protocol. This structure contains a pointer to the **RasEapEnd** function.

Provided that **RasEapBegin** returned successfully, the Connection manager calls the **RasEapEnd** function when authentication has completed.

! Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

+ See Also

Extensible Authentication Protocol Reference, EAP Functions, **RasEapBegin**, **RasEapGetInfo**, **PPP_EAP_INFO**, **PPP_EAP_INPUT**

RasEapFreeMemory

The Connection Manager calls **RasEapFreeMemory** to free memory buffers returned by **RasEapInvokeConfigUI**, **RasEapGetIdentity**, and **RasEapInvokeInteractiveUI**.

```
DWORD RasEapFreeMemory(
  BYTE * pMemory    // pointer to the memory to free
);
```

Parameters

pMemory
Pointer to the memory to free.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h.

Remarks

An authentication protocol may implement its various user interfaces in different DLLs. In such a case, each DLL must implement the **RasEapFreeMemory** function.

It is also possible that a single DLL may implement multiple user interfaces. For example, a single DLL may implement both the configuration and identity user interface for an authentication protocol. Another example would be a DLL that implements two configuration user interfaces, each to support a different authentication protocol. In these cases, the DLL must implement a single version of **RasEapFreeMemory** that can free memory returned from any of the user interfaces implemented in the DLL.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Functions, **RasEapInvokeConfigUI**, **RasEapGetIdentity**, **RasEapInvokeInteractiveUI**

RasEapGetIdentity

The RAS Connection Manager calls the **RasEapGetIdentity** function to obtain identity information for the user requesting authentication.

```
DWORD RasEapGetIdentity(
  DWORD dwEapTypeId,             // identifies the protocol
  HWND  hwndParent,              // handle to parent window
  DWORD dwFlags,                 // flags that qualify
```

```
                                     // authentication process
    const WCHAR *  pwszPhonebook,    // path to phone book to
                                     // use
    const WCHAR *  pwszEntry,        // name of entry in phone
                                     // book
    BYTE * pConnectionDataIn,        // pointer to current
                                     // connection data
    DWORD  dwSizeOfConnectionDataIn, // size of current
                                     // connection data
    BYTE * pUserDataIn,              // pointer to current
                                     // user data from registry
    DWORD  dwSizeOfUserDataIn,       // size of current user
                                     // data
    BYTE * * ppUserDataOut,          // new user data
    DWORD *  pdwSizeOfUserDataOut,   // size of new user data
    WCHAR *  ppwszIdentity           // identity of user
);
```

Parameters

dwEapTypeId

Specifies the authentication protocol for which to invoke the identity user interface.

hwndParent

Handle to the parent window for the user interface dialog. If the *dwFlags* parameter contains the RAS_EAP_FLAG_NON_INTERACTIVE flag, then *hwndParent* is NULL.

dwFlags

Specifies zero or more of the following flags that qualify the authentication process.

Flag	Description
RAS_EAP_FLAG_ROUTER	Specifies that the computer that is dialing in is a router. The absence of this flag indicates that the computer dialing in is a RAS client.
RAS_EAP_FLAG_NON_INTERACTIVE	Specifies that the authentication protocol should not bring up a user-interface. If the authentication protocol is not able to determine the identity from the data supplied, it should return an error. If this flag is specified, the *hwndParent* parameter will be NULL.
RAS_EAP_FLAG_LOGON	Specifies that the user data is obtained from Winlogon.
RAS_EAP_FLAG_PREVIEW	Specifies that the user should be prompted for identity information before dialing.

(continued)

(continued)

Flag	Description
RAS_EAP_FLAG_FIRST_LINK	Indicates that this connection is the first link in a multilink connection. See Multilink and Callback Connections for more information.

pwszPhonebook
Pointer to a Unicode string that contains the full path of the Phone-Book (PBK) file. If this parameter is NULL, the function uses the system phone book.

pwszEntry
Pointer to a Unicode string that contains an existing entry name.

pConnectionDataIn
Pointer to the connection-specific data currently stored in the phone-book entry.

dwSizeOfConnectionDataIn
Size of the connection-specific data currently stored in the phone-book entry.

pUserDataIn
Pointer to the user-specific data currently stored for this user in the registry.

dwSizeOfUserDataIn
Specifies the size of the user-specific data currently stored for this user in the registry.

ppUserDataOut
Pointer to a pointer that, on successful return, points to the identity data for the user. This data will be passed to the authentication protocol in the **pUserData** member of **PPP_EAP_INPUT** during the call to **RasEapBegin**.

The authentication protocol should allocate the memory buffer for the identity data. RAS will free this memory by calling **RasEapFreeMemory**.

pdwSizeOfUserDataOut
Pointer to a **DWORD** value that, on successful return, contains the size of the data pointed to by the *ppUserDataOut* parameter.

ppwszIdentity
Pointer to a pointer that, on successful return, points to a Unicode string that identifies the user requesting authentication. This string will be passed to the authentication protocol in the **pszIdentity** member of **PPP_EAP_INPUT** during the call to **RasEapBegin**.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function was not able to allocate memory for the user data, the return value should be ERROR_NOT_ENOUGH_MEMORY.

If the function is called with the RAS_EAP_FLAG_NON_INTERACTIVE flag, but must invoke a user interface to determine the user's identity, the function should return ERROR_INTERACTIVE_MODE.

If the function fails in some other way, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h.

Remarks

The DLL that implements **RasEapGetIdentity** and **RasEapFreeMemory** may support more than one authentication protocol. The *dwEapTypeId* parameter specifies for which protocol to invoke the identity user interface.

The authentication protocol receives the data returned from **RasEapGetIdentity** in the **pUserData** member of **PPP_EAP_INPUT** during **RasEapBegin**. To store the data for this user in the registry, the authentication protocol should set the **pUserData** member of **PPP_EAP_OUTPUT** to point to the data, and the **fSaveUserData** member of **PPP_EAP_OUTPUT** to TRUE.

This function is called by the RAS function, **RasGetEapUserIdentity**.

If **RasEapGetIdentity** displays a user interface, the user interface must support **WM_COMMAND** messages where **LOWORD**(*wParam*) equals IDCANCEL.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Functions, Obtaining Identity Information, **RasEapFreeMemory**, **RasEapMakeMessage**, **RasGetEapUserIdentity**, **PPP_EAP_INPUT**

RasEapGetInfo

The Connection Manager calls **RasEapGetInfo** to obtain a set of function pointers for a specified authentication protocol.

```
DWORD RasEapGetInfo(
  DWORD dwEapTypeId            // identifies the protocol
  PPP_EAP_INFO * pEapInfo      // pointer to information for
                               // a particular EAP
);
```

Parameters

dwEapTypeId
Specifies the authentication protocol for which to obtain information.

pEapInfo
Pointer to a **PPP_EAP_INFO** structure. The structure contains members that RAS sets to identify the structure version and the authentication protocol for which function pointers are requested. For more information, see ***PPP_EAP_INFO***.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h.

Remarks

The DLL that implements **RasEapGetInfo** may support more than one authentication protocol. The *dwEapTypeId* parameter specifies for which authentication protocol to obtain information.

Implementations of EAP must export the **RasEapGetInfo** function, since RAS uses **RasEapGetInfo** to obtain pointers to the other authentication protocol functions.

Upon initialization, the Connection Manager calls **RasEapGetInfo** for each EAP DLL installed in the registry subkey, as explained in the EAP Overview.

If the function returns any value other than **NO_ERROR**, RAS considers the authentication protocol to be non-functional. RAS posts an error to the Microsoft® Windows NT®/Windows® 2000 Event Log indicating that this protocol did not start correctly and therefore is not available.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Functions, EAP (Extensible Authentication Protocol) Overview, **PPP_EAP_INFO**

RasEapInitialize

The RAS Connection Manager calls the **RasEapInitialize** function to initialize or deinitialize the authentication protocol.

```
DWORD RasEapInitialize(
  BOOL fInitialize    // TRUE to init, FALSE to deinit
);
```

Parameters

fInitialize

Specifies whether the authentication protocol should initialize or deinitialize. This parameter is TRUE if the protocol should initialize and FALSE if the protocol should deinitialize.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h.

Remarks

The **RasEapInitialize** function is not part of the RRAS API; it is implemented in the EAP DLL. When the Connection Manager calls the **RasEapGetInfo** function, it receives a **PPP_EAP_INFO** structure for the authentication protocol. This structure contains a pointer to the **RasEapInitialize** function.

The authentication protocol may set the **RasEapInitialize** member in **PPP_EAP_INFO** to NULL. A NULL value indicates that the authentication protocol does not require initialization or deinitialization. Therefore, RAS need not call this function.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Functions, **PPP_EAP_INFO**, **RasEapBegin**, **RasEapGetInfo**

RasEapInvokeConfigUI

The Connection Manager calls the **RasEapInvokeConfigUII** function to display a dialog to obtain configuration information from the user. RAS calls **RasEapInvokeConfigUI** when a new phone-book entry is created or an existing phone-book entry is edited, provided that the authentication protocol for the entry provides a configuration user interface.

```
DWORD RasEapInvokeConfigUI(
  DWORD dwEapTypeId,          // identifies the protocol
  HWND hwndParent,            // handle to parent window
  DWORD dwFlags,              // indicates whether
```

(continued)

(continued)

```
                                    // caller is router or RAS
                                    // client
  BYTE * pConnectionDataIn,         // current connection data
  DWORD dwSizeOfConnectionDataIn,   // size of current
                                    // connection data
  BYTE * * ppConnectionDataOut,     // new connection data
  DWORD *  pdwSizeOfConnectionDataOut,  // size of new
                                        // connection data
);
```

Parameters

dwEapTypeId
Specifies the authentication protocol for which to invoke the configuration UI.

hwndParent
Handle to the parent window for the UI dialog.

dwFlags
Specifies whether the computer that is dialing in is a router or a RAS client. If the computer is a router, this parameter should be set to:

RAS_EAP_FLAG_ROUTER

Otherwise, this parameter should be zero.

pConnectionDataIn
Pointer to the connection data currently stored in the phone-book entry. If the phone-book entry does not contain any data, this parameter is NULL.

dwSizeOfConnectionDataIn
Specifies the size of the connection data currently stored in the phone-book entry. If the phone-book entry for this connection does not contain any data, this parameter will be zero.

ppConnectionDataOut
Pointer to a pointer that, on successful return, points to the new connection data to store in the phone-book entry. None of this data should be specific to the current machine; phone-book entries should be portable from machine to machine.

pdwSizeOfConnectionDataOut
Pointer to a **DWORD** that, on successful return, points to the size of the new connection data to store in the phone-book entry.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function was not able to allocate memory for the configuration data, the return value should be ERROR_NOT_ENOUGH_MEMORY.

If the function fails in some other way, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h.

Remarks

The DLL that implements **RasEapInvokeConfigUI** and **RasEapFreeMemory** may support more than one authentication protocol. The *dwEapTypeId* parameter specifies for which protocol to invoke the configuration UI.

RAS stores the connection data returned by **RasEapInvokeConfigUI** in the phone-book entry for the connection on the client computer.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Functions, Client-Side Configuration User Interface,

RasEapFreeMemory,

RasEapGetIdentity, **RasEapInvokeInteractiveUI**

RasEapInvokeInteractiveUI

The RAS Connection Manager calls the **RasEapInvokeInteractiveUI** function to display a dialog to obtain authentication data from the user.

```
DWORD RasEapInvokeInteractiveUI(
  DWORD dwEapTypeId,          // identifies the protocol
  HWND hwndParent,            // handle to parent window
  BYTE * pUIContextData,      // pointer to context data
  DWORD dwSizeofUIContextData,        // size of context data
  BYTE * * ppDataFromInteractiveUI,  // pointer to data
                                      // returned from UI
  DWORD *  pdwSizeOfDataFromInteractiveUI // size of data
                                      // returned from UI.
);
```

Parameters

dwEapTypeId
: Identifies the authentication protocol for which to invoke the interactive UI.

hwndParent
: Handle to the parent window for the dialog.

pUIContextData
Pointer to context data for the interactive UI. The authentication protocol provides a pointer to this data as a member of the **PPP_EAP_OUTPUT** structure. The RAS Connection Manager receives the **PPP_EAP_OUTPUT** structure as an output parameter from the **RasEapMakeMessage** function.

dwSizeofUIContextData
Specifies the size of the context data. The authentication protocol provides the size as a member of the **PPP_EAP_OUTPUT** structure. The RAS Connection Manager receives the **PPP_EAP_OUTPUT** structure as an output parameter from the **RasEapMakeMessage** function.

ppDataFromInteractiveUI
Pointer to a pointer variable. On successful return, this pointer variable will point to a memory buffer that contains the data obtained by the interactive UI. The interactive UI allocates this memory. RAS passes this data back to the authentication protocol in the **PPP_EAP_INPUT** structure, then RAS frees this memory by calling **RasEapFreeMemory**.

If the interactive UI does not obtain any user-specific data, the pointer that *ppUserData* points to should be set to NULL.

pdwSizeOfDataFromInteractiveUI
Pointer to a **DWORD** variable to receive the size of the data returned from the interactive UI. If the interactive UI does not obtain any user-specific data, the **DWORD** variable should be set to zero.

Return Values

If the function succeeds, the return value is NO_ERROR. Check the *ppDataFromInteractiveUI and lpdwSizeOfDataFromInteractiveUI* parameters to determine if the function returned data from the interactive UI.

If the function was not able to allocate memory for the data, the return value should be ERROR_NOT_ENOUGH_MEMORY.

If the function fails in some other way, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h.

Remarks

The DLL that implements the **RasEapInvokeInteractiveUI** and **RasEapFreeMemory** functions may support more than one authentication protocol. The *dwEapTypeId* parameter specifies the authentication protocol for which to invoke the interactive UI.

A pointer to the data returned from the interactive UI is passed back to the authentication protocol in the **pDataFromInteractiveUI** member of **PPP_EAP_INPUT** structure. The **PPP_EAP_INPUT** structure is passed as a parameter to the **RasEapMakeMessage** function.

The interactive user interface must support **WM_COMMAND** messages where **LOWORD**(*wParam*) equals IDCANCEL.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Functions, Interactive User Interface, **RasEapFreeMemory**, **RasEapInvokeConfigUI**, **RasEapGetIdentity**, **RasEapMakeMessage**, **PPP_EAP_INPUT**, **PPP_EAP_OUTPUT**

RasEapMakeMessage

The **RasEapMakeMessage** function is the framework within which incoming and outgoing EAP packets, time outs, and other events such as authentication completion are processed for an EAP. RAS calls the **RasEapMakeMessage** function every time there is an incoming or outgoing packet.

```
DWORD ( * RasEapMakeMessage ) (
  VOID * pWorkBuf,      // pointer to the workbuffer for this
                        // authentication sesssion
  PPP_EAP_PACKET * pReceivePacket,  // pointer to incoming
                                    // packet
  PPP_EAP_PACKET * pSendPacket,     // pointer to packet to
                                    // transmit
  DWORD cbSendPacket,               // max size of packet to
                                    // transmit
  PPP_EAP_OUTPUT * pEapOutput,      // info/requests from
                                    // EAP to RAS
  PPP_EAP_INPUT * pEapInput         // info/requests from
                                    // RAS to EAP
);
```

Parameters

pWorkBuf

Pointer to the work buffer. The authentication protocol provides RAS with a pointer to this buffer via the **RasEapBegin** function.

pReceivePacket

Pointer to a **PPP_EAP_PACKET** structure that contains a received packet. A *pReceivePacket* value of NULL indicates either that RAS is initiating the dialog with the authentication protocol, or that a time out has occurred and the authentication protocol should resend the last packet. The authentication protocol must determine, based on context, which of these two cases is true.

pSendPacket
Pointer to a **PPP_EAP_PACKET** structure. The authentication protocol can use this structure to specify a packet to send.

cbSendPacket
Specifies the size, in bytes, of the buffer pointed to by *pSendPacket*.

pEapOutput
Pointer to **PPP_EAP_OUTPUT** structure.

pEapInput
Pointer to a **PPP_EAP_INPUT** structure. This parameter may be NULL.

Return Values

If the function succeeds, the return value is NO_ERROR.

If the function fails, the return value should be an appropriate error code from Winerror.h, Raserror.h, or Mprerror.h. Any error except for ERROR_PPP_INVALID_PACKET, terminates the authentication session. For more information on the ERROR_PPP_INVALID_PACKET return code, see *EAP Implementation Details*.

Remarks

The **RasEapMakeMessage** function is not part of the RRAS API; it is implemented in the EAP DLL. When the Connection Manager calls the **RasEapGetInfo** function, it receives a **PPP_EAP_INFO** structure for the authentication protocol. This structure contains a pointer to the **RasEapMakeMessage** function.

RAS allocates the buffers pointed to by *pReceivePacket*, *pSendPacket*, *pEapOutput*, and *pEapInput*. The authentication protocol does not allocate any of these buffers.

! Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

+ See Also

Extensible Authentication Protocol Reference, EAP Functions, **RasEapGetInfo**, **PPP_EAP_INFO**, **PPP_EAP_INPUT**, **PPP_EAP_OUTPUT**, **PPP_EAP_PACKET**

EAP Structures

Vendors should use the following structure types for authentication protocols and authentication providers.

PPP_EAP_INFO
PPP_EAP_INPUT
PPP_EAP_OUTPUT
PPP_EAP_PACKET
RAS_AUTH_ATTRIBUTE

PPP_EAP_INFO

The **PPP_EAP_INFO** structure provides the Connection Manager with information about the authentication protocol, including pointers to functions located in the EAP DLL.

```
typedef struct _PPP_EAP_INFO {
  DWORD dwSizeInBytes;  // size of struct identifies version
  DWORD dwEapTypeId;    // identifies the authentication protocol
  DWORD ( * RasEapInitialize) (
      BOOL              fInitialize
  );
  DWORD ( * RasEapBegin ) (
      VOID * *          ppWorkBuffer,
      PPP_EAP_INPUT *   pPppEapInput
  );
  DWORD ( * RasEapEnd ) (
      VOID *            pWorkBuffer
  );
  DWORD ( * RasEapMakeMessage ) (
      VOID*             pWorkBuf,
      PPP_EAP_PACKET*   pReceivePacket,
      PPP_EAP_PACKET*   pSendPacket,
      DWORD             cbSendPacket,
      PPP_EAP_OUTPUT*   pEapOutput,
      PPP_EAP_INPUT*    pEapInput
  );
} PPP_EAP_INFO, *PPPP_EAP_INFO;
```

Members

dwSizeInBytes

Specifies the size of the **PPP_EAP_INFO** structure. RAS passes this value to the EAP DLL. The DLL uses this value to determine which version of the **PPP_EAP_INFO** structure RAS is using.

dwEapTypeId
Specifies a particular authentication protocol. This identifier must be unique throughout industry-wide implementation of EAP (see *IETF Internet Draft 1310*). The implementer of an authentication protocol must obtain this identifier from the Internet Assigned Numbers Authority (IANA).

(* **RasEapInitialize**)
Pointer to the **RasEapInitialize** function for the authentication protocol. The authentication protocol sets the value of this member. The authentication protocol may set this member to NULL, in which case the protocol does not require RAS to call this function.

(* **RasEapBegin**)
Pointer to the **RasEapBegin** function for the requested authentication protocol. The authentication protocol sets the value of this member. This member may be NULL, in which case, the authentication protocol does not require any initialization. If this member is NULL, RAS ignores the **RasEapEnd** member.

(* **RasEapEnd**)
Pointer to the **RasEapEnd** function for the authentication protocol. The authentication protocol sets the value of this member.

(* **RasEapMakeMessage**)
Pointer to the **RasEapMakeMessage** for the requested authentication protocol. The authentication protocol sets the value of this member.

Remarks

A given EAP DLL may implement more than one authentication protocol. Use the **dwEapTypeId** member to specify for which protocol to retrieve information.

! Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

+ See Also

Extensible Authentication Protocol Reference, EAP Structures, **RasEapBegin**, **RasEapEnd**, **RasEapGetInfo**, **RasEapMakeMessage**.

PPP_EAP_INPUT

The **PPP_EAP_INPUT** structure is used in the interaction between the RAS Connection Manager Service PPP implementation and the EAP to provide user information, and to facilitate the use of authentication providers such as Windows 2000 domain authentication or RADIUS.

```
typedef struct _PPP_EAP_INPUT {
  DWORD     dwSizeInBytes   // size of this structure
  DWORD     fFlags          // flags that qualify the
                            // authentication process
  BOOL      fAuthenticator; // act as authenticator or
                            // authenticatee
  WCHAR *   pwszIdentity;   // users's identity
  WCHAR *   pwszPassword;   // user's account password.
  BYTE      bInitialId;     // ID of initial EAP packet
  RAS_AUTH_ATTRIBUTE * pUserAttributes;
  BOOL      fAuthenticationComplete;
  DWORD     dwAuthResultCode;
  HANDLE   hTokenImpersonateUser // handle to impersonate
                                 // user being authenticated
  DWORD    fSuccessPacketReceived  // true if success
                                   // indicated by NCP
                                   // packet
  DWORD    fDataReceivedFromInteractiveUI  // true if user
                                           // exits from
                                           // interactive UI
  PBYTE    pDataFromInteractiveUI          // pointer to
                                           // data from the
                                           // interactive UI
  DWORD    dwSizeOfDataFromInteractiveUI   // size of data
                                           // from the
                                           // interactive UI
  PBYTE    pConnectionData                 // pointer to
                                           // connection-
                                           // specific data
                                           // from config UI
  DWORD    dwSizeOfConnectionData          // size of
                                           // connection-
                                           // specific data
                                           // from config UI
  PBYTE    pUserData                       // pointer to
                                           // user data from
                                           // identity UI
  DWORD    dwSizeOfUserData                // size of user
                                           // data from
                                           // identity UI
  HANDLE   hReserved                       // reserved
} PPP_EAP_INPUT, * PPPP_EAP_INPUT;
```

Members

dwSizeInBytes

Specifies the size in bytes of the **PPP_EAP_INPUT** structure. The value of this member can be used to distinguish between current and future versions of this structure.

fFlags

Specifies zero or more of the following flags that qualify the authentication process.

Flag	Description
RAS_EAP_FLAG_ROUTER	Specifies whether the computer dialing in is a router or a RAS client. If the computer is a router, this parameter should be set.
RAS_EAP_FLAG_NON_INTERACTIVE	Specifies that the authentication protocol should not bring up a user-interface. If the authentication protocol is not able to determine the identity from the data supplied, it should return an error.
RAS_EAP_FLAG_LOGON	Specifies that the user data from obtained from Winlogon.
RAS_EAP_FLAG_FIRST_LINK	Indicates that this connection is the first link in a multilink connection. See Multilink and Callback Connections for more information.

fAuthenticator

Specifies whether the authentication protocol is operating on the server or client. A value of TRUE indicates that the authentication protocol is operating on the server as the authenticator. A value of FALSE indicates that the authentication protocol is operating on the client as the as the process to be authenticated.

pwszIdentity

Pointer to an Unicode that identifies the user requesting authentication. This string is of the form domain\user or machine\user.

If the authentication protocol is able to derive the user's identity from an additional source, for example a certificate, it should verify that the identity so derived matched the value of **pszIdentity**.

pwszPassword

Pointer to a Unicode string that contains the user's account password. Available only if **fAuthenticator** is FALSE. This member may be NULL.

bInitialId

Specifies the identifier of the initial EAP packet sent by the DLL. This value is incremented by one for each subsequent request packet.

pUserAttributes

Pointer to an array of **RAS_AUTH_ATTRIBUTE** structures. The array is terminated by a structure with an **raaType** member that has a value of *raatMinimum* (see ***RAS_AUTH_ATTRIBUTE_TYPE***) During the RasEapBegin call, this array contains attributes that describe the currently dialed-in user. When the **fAuthenticationComplete** member is TRUE, this array may contain attributes returned by the authentication provider.

fAuthenticationComplete

Specifies a Boolean value indicating whether the authentication provider has authenticated the user. A value of TRUE indicates authentication completion. Check the *dwAuthResultCode* field to determine if the authentication was successful. Ignore this field if the authentication protocol is not using an authentication provider.

dwAuthResultCode

Specifies the result of the authentication provider's authentication process. Successful authentication results in NO_ERROR. Authentication failure codes for **dwAuthResultCode** must come only from Winerror.h, Raserror.h or Mprerror.h. Ignore this field if the authentication protocol is not using an authentication provider.

hTokenImpersonateUser

Handle to an impersonation token for the user requesting authentication. This member is valid only on the client side. For more information on impersonation tokens, see *Access Tokens*.

fSuccessPacketReceived

RAS sets this member to TRUE if the client receives an Network Control Protocol (NCP) packet even though the client has not yet received an EAP success packet. The EAP success packet is a non-acknowledged packet. Therefore, it may be lost and not resent by the server. In this situation, the receipt of an NCP packet indicates that authentication must have been successful, since the server has moved on to the NCP phase of PPP. This member should be examined only on the client side.

fDataReceivedFromInteractiveUI

RAS sets this member to TRUE whenever the user exits from the authentication protocol's interactive user interface.

pDataFromInteractiveUI

Pointer to data received from the authentication protocol's interactive user interface. This pointer is non-NULL if the **fDataReceivedFromInteractiveUI** member is TRUE and the interactive user interface did, in fact, return data. Otherwise, this pointer is NULL.

If non-NULL, the authentication protocol should make a copy of the data in its own memory space. RAS frees the memory occupied by this data on return from the call in which the **PPP_EAP_INPUT** structure was passed. To free the memory, RAS calls the **RasEapFreeMemory** function.

dwSizeOfDataFromInteractiveUI

Specifies the size, in bytes, of the data pointed to by **pDataFromInteractiveUI**. If no data is returned from the interactive user interface, this member is zero.

pConnectionData

Pointer to connection data received from the authentication protocol's configuration user interface. This data is available only when the **PPP_EAP_INPUT** structure is passed in **RasEapBegin**. It is not available in calls to **RasEapMakeMessage**.

The authentication protocol should make a copy of this data in its own memory space. RAS frees the memory occupied by this data on return from the call in which the **PPP_EAP_INPUT** structure was passed. To free the memory, RAS calls the **RasEapFreeMemory** function.

If the authentication protocol's configuration user interface does not return any data, this member is NULL.

dwSizeOfConnectionData

Specifies the size in bytes of the data pointed to by **pConnectionData**. If **pConnectionData** is NULL, this member is zero.

pUserData

Pointer to user data received from the authentication protocol's **RasEapGetIdentity** function on the client computer. If the authentication protocol does not implement **RasEapGetIdentity**, this member points to data from the registry for this user.

This data is available only when the **PPP_EAP_INPUT** structure is passed in **RasEapBegin**. It is not available in calls to **RasEapMakeMessage**.

The authentication protocol should make a copy of this data in its own memory space. RAS frees the memory occupied by this data on return from the call in which the **PPP_EAP_INPUT** structure was passed.

If the **RasEapGetIdentity** function is not implemented or did not return any data, and no data exists for the user in the registry, this member is NULL.

dwSizeOfUserData

Specifies the size, in bytes, of the data pointed to by **pUserData**. If **pUserData** is NULL, this member is zero.

hReserved

This member is reserved.

Remarks

The PPP_EAP_INPUT structure is passed by RAS to the authentication protocol in calls to **RasEapBegin** and **RasEapMakeMessage**.

The **pszIdentity** and **pszPassword** members of the **PPP_EAP_INPUT** structure are used by the **RasEapBegin** function to obtain user information. The **pszPassword** member is non-NULL only if the **fAuthenticator** member is FALSE, that is, the authentication protocol is running on the client computer.

If the authentication protocol is using an authentication provider, such as Radius or Windows 2000 domain authentication, the following members are used to interface with the authentication provider:

pUserAttributes
fAuthenticationComplete
dwAuthResultCode

Note that the array of **RAS_AUTH_ATTRIBUTE** structures is passed only if **fAuthenticator** is TRUE. This array contains current session information such as port identifier and local IP address.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Structures, **RasEapBegin**, **RasEapGetIdentity**, **RasEapFreeMemory**, **RasEapMakeMessage**, **RAS_AUTH_ATTRIBUTE**

PPP_EAP_OUTPUT

The authentication protocol uses the **PPP_EAP_OUTPUT** structure to communicate requests and status information to the Connection Manager on return from calls to **RasEapMakeMessage**.

```
typedef struct _PPP_EAP_OUTPUT {
  PPP_EAP_ACTION        Action;              // action that RAS
                                             // should take
  DWORD                 dwAuthResultCode;    // result of
                                             // authentication
  RAS_AUTH_ATTRIBUTE * pUserAttributes;      // array of
                                             // attributes
                                             // structures
  BOOL                  fInvokeInteractiveUI;   // causes RAS
                                                // to invoke
                                                // interactive
                                                // UI
  PBYTE                 pUIContextData;         // data to
                                                // send to
                                                // interactive
                                                // UI
  DWORD                 dwSizeOfUIContextData;  // size of
                                                // data
  BOOL              fSaveConnectionData;
```

(continued)

(continued)

```
    PBYTE           pConnectionData;
    DWORD           dwSizeOfConnectionData;
    BOOL            fSaveUserData;
    PBYTE           pUserData;          // pointer to the
                                        // user
                                        // identity info
    DWORD           dwSizeofUserData;   // size of the user
                                        // identity info
} PPP_EAP_OUTPUT, *PPPP_EAP_OUTPUT;
```

Members

Action

Specifies a **PPP_EAP_ACTION** value. The Connection Manager carries out this action on behalf of the authentication protocol.

dwAuthResultCode

Specifies whether authentication was successful. Any non-zero value for **dwAuthResultCode** indicates failure. The failure code must come from Winerror.h, Raserror.h or Mprerror.h. This member is valid only if the **Action** member has a value of **EAPACTION_Done** or **EAPACTION_SendAndDone**.

pUserAttributes

Pointer to an optional array of **RAS_AUTH_ATTRIBUTE** structures. The array is terminated by a structure with an **raaType** member that has a value of *raatMinimum* (see ***RAS_AUTH_ATTRIBUTE_TYPE***).

This member should be set on the authenticator side when **Action** is EAPACTION_Authenticate, or when **Action** is EAPACTION_Done or EAPACTION_SendAndDone. and **dwAuthResultCode** is zero.

When **Action** is EAPACTION_Authenticate, the array may contain additional attributes necessary to authenticate the user, e.g. the user-password. If the authentication protocol passes in only the user name, RAS does not invoke the authentication provider to authenticate the user, Instead, RAS just passes back the current attributes for the user.

When **Action** is EAPACTION_Done or EAPACTION_SendAndDone, and **dwAuthResultCode** is zero, the array may contain additional attributes to assign to the user. These attributes overwrite any attributes of the same type returned by the authentication provider.

The authentication protocol should free this memory in its **RasEapEnd** function.

fInvokeInteractiveUI

Specifies whether RAS should invoke the authentication protocol's interactive UI. If the authentication protocol sets this member to TRUE, RAS invokes the interactive UI, by calling the **RasEapInvokeInteractiveUI** function provided by the authentication protocol.

pUIContextData
Pointer to context data that RAS should pass in the call to **RasEapInvokeInteractiveUI**. The authentication protocol should free this memory in its implementation of **RasEapEnd**.

dwSizeOfUIContextData
Specifies the size of the context data that RAS should pass in the call to **RasEapInvokeInteractiveUI**.

fSaveConnectionData
Specifies whether RAS should save the information pointed to by the **pConnectionData** member. If **fSaveConnectionData** is TRUE, RAS will save the data in the phone book. This is only valid for the process that is being authenticated.

pConnectionData
Specifies data specific to the connection, that is, data that is not specific to any particular user. If the **fSaveConnectionData** member is TRUE, RAS saves the connection data in the phone book. The authentication protocol should free the memory occupied by this data during the call to **RasEapEnd**.

dwSizeOfConnectionData;
Specifies the size, in bytes, of the data pointed to by the **pConnectionData** member.

fSaveUserData
Specifies whether RAS should save the user data pointed to by the **pUserData** member. If this parameter is TRUE, RAS saves the user-specific data in the registry under HKEY_CURRENT_USER.

pUserData
Pointer to user data that RAS should save in the registry. RAS saves this data in the registry under HKEY_CURRENT_USER. The authentication protocol should free this memory during the call to **RasEapEnd**.

dwSizeofUserData
Specifies the size in bytes of the data pointed to by **pUserData**.

Remarks

Use the **RasEapMakeMessage** function to pass the **PPP_EAP_OUTPUT** structure between the authentication protocol and the Connection Manager

The authentication protocol may use the **PPP_EAP_OUTPUT** structure to return the Microsoft Point to Point Encryption (MPPE) session key. The authentication protocol should place the session key in the value field of a *sub*-attribute contained within the value field of an attribute of type raatVendorSpecific (see ***RAS_AUTH_ATTRIBUTE_TYPE***). The sub-attribute should have a Vendor-ID of 311 (Microsoft) and a Vendor-Type of 12 (MS-CHAP-MPPE-Keys). The authentication protocol should set the **pUserAttributes** member to point to the raatVendorSpecific attribute, and set the **Action** member to EAPACTION_Done or EAPACTION_SendAndDone. For more information about the format of the MPPE sub-attribute see *http://search.ietf.org/internet-drafts/draft-ietf-radius-mschap-attr-01.txt*.

For more information about attribute formats see ***RAS_AUTH_ATTRIBUTE***, ***RAS_AUTH_ATTRIBUTE_TYPE***, and *http://src.doc.ic.ac.uk/computing/internet/rfc/rfc2138.txt*.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Structures, **RAS_AUTH_ATTRIBUTE**, **PPP_EAP_ACTION**, **RasEapInvokeInteractiveUI**, **RasEapMakeMessage**

PPP_EAP_PACKET

The **PPP_EAP_PACKET** structure specifies information about a packet being processed by the authentication protocol.

```
typedef struct _PPP_EAP_PACKET {
  BYTE    Code;        // 1-Request, 2-Response,
                       // 3-Success, 4-Failure
  BYTE    Id;          // Id of this packet
  BYTE    Length[2];   // Length of this packet
  BYTE    Data[1];     // Data, First byte is Type for
                       // Request/Response
} PPP_EAP_PACKET, * PPPP_EAP_PACKET;
```

Members

Code

Specifies the type of packet that is being sent or received by the authentication protocol. This parameter can be one of the four following values.

Value	Meaning
EAPCODE_Request	The packet is a request.
EAPCODE_Response	The packet is a response.
EAPCODE_Success	The packet indicates success.
EAPCODE_Failure	The packet indicates failure.

Id

Specifies the identifier of the packet. The authentication protocol is responsible for maintaining packet counts for sessions, as that packet count pertains to EAP activity.

Length[2]

Specifies the length of the packet.

Data[1]

Specifies the data transmitted by this packet. If the packet is a request or a response packet, the first byte of this member signifies its type. For more information about packet types and requirements for type reservation, refer to the PPP EAP Internet draft, found at http://ds2.internic.net/internet-drafts/draft-ietf-pppext-eap-auth-02.txt.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Structures, **RasEapGetInfo**, **RasEapMakeMessage**, **PPP_EAP_INFO**, **PPP_EAP_INPUT**, **PPP_EAP_OUTPUT**

RAS_AUTH_ATTRIBUTE

The **RAS_AUTH_ATTRIBUTE** structure is used to pass authentication attributes, of type **RAS_AUTH_ATTRIBUTE_TYPE**, during an EAP session.

```
typedef struct _RAS_AUTH_ATTRIBUTE {
  RAS_AUTH_ATTRIBUTE_TYPE raaType;   // attribute type
  DWORD                   dwLength;  // length of Value
  PVOID                   Value;     // pointer to value or
                                     // contains value
}RAS_AUTH_ATTRIBUTE, *PRAS_AUTH_ATTRIBUTE;
```

Members

raaType

Specifies the type of attribute, as defined in the **RAS_AUTH_ATTRIBUTE_TYPE** enumerated type.

dwLength

Specifies the length in bytes of the value of this attribute. If the **Value** member is a pointer, **dwLength** specifies the length of the buffer pointed to. If the **Value** member is the value itself, **dwLength** specifies how much of the length of the **Value** member is taken up by the value.

Value

Specifies the value of the attribute. Although this member is of the **PVOID** type, this member sometimes contains the value of the attribute rather than pointing to the value. The only way to know whether to interpret the **Value** member as a pointer to the value or the value itself, is to check the **raaType** member. See the reference page for **RAS_AUTH_ATTRIBUTE_TYPE** for information about how the **Value** member should be interpreted for different types.

Remarks

Often an array of these structures is used to store or obtain a set of attributes for a given user. Since the number of attributes for a session is unknown, the array must be dynamic. The array is terminated by a structure with an **raaType** member that has a value of *raatMinimum*

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Structures, **RAS_AUTH_ATTRIBUTE_TYPE**

EAP Enumerated Types

Use the following enumerated types for authentication protocols and authentication providers:

PPP_EAP_ACTION

RAS_AUTH_ATTRIBUTE_TYPE

PPP_EAP_ACTION

The **PPP_EAP_ACTION** enumerated type specifies actions that the Connection Manager should take on behalf of the authentication protocol.

```
typedef enum _PPP_EAP_ACTION {
  EAPACTION_NoAction,        // Be passive, i.e., listen
                             // without timeout (default)
  EAPACTION_Authenticate,    // invoke the authentication
                             // provider
  EAPACTION_Done,            // End auth session,
                             // dwAuthResultCode is set
  EAPACTION_SendAndDone,     // As above but send message
                             // without timeout first
  EAPACTION_Send,            // Send message, don't timeout
                             // waiting for reply
  EAPACTION_SendWithTimeout, // Send message, timeout if
                             // reply not received
  EAPACTION_SendWithTimeoutInteractive // As above, but
                                       // don't increment
                                       // retry count
} PPP_EAP_ACTION;
```

Values

EAPACTION_NoAction
Directs the Connection Manager to be passive.

EAPACTION_Done
Directs the Connection Manager Service to end the authentication session. EAPACTION_Done indicates that the **dwAuthResultCode** member of the **PPP_EAP_OUTPUT** structure is set with an appropriate value.

EAPACTION_SendAndDone
Directs the Connection Manager to send a message (without a time out), then end the authentication session. EAPACTION_SendAndDone indicates that the **dwAuthResultCode** member of the **PPP_EAP_OUTPUT** structure is set with an appropriate value.

EAPACTION_Send
Directs the Connection Manager to send a message without setting a time out to wait for a reply.

EAPACTION_SendWithTimeout
Directs the Connection Manager to send a message and set a time out to wait for a reply.

EAPACTION_SendWithTimeoutInteractive
Directs the Connection Manager to send a message and set a time out to wait for a reply, but instructs the Connection Manager not to increment the retry counter.

EAPACTION_Authenticate
Directs the Connection Manager to invoke the authentication provider to authenticate the user.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

See Also

Extensible Authentication Protocol Reference, EAP Enumerated Types, **PPP_EAP_INPUT**, **PPP_EAP_OUTPUT**

RAS_AUTH_ATTRIBUTE_TYPE

The **RAS_AUTH_ATTRIBUTE_TYPE** enumerated type specifies attribute values used for session authentication. Further details for values in this enumerated type may be obtained by referring to one of the three following references: *RFC 2138, RFC 2139,* or *draft-ietf-radius-ext-04*.

```
typedef enum _RAS_AUTH_ATTRIBUTE_TYPE_ {
  raatMinimum = 0,            // Undefined
  raatUserName,               // Value field is a pointer
  raatUserPassword,           // Value field is a pointer
  raatMD5CHAPPassword,        // Value field is a pointer
  raatNASIPAddress,           // Value field is a 32 bit
                              // integral value
  raatNASPort,                // Value field is a 32 bit
                              // integral value
  raatServiceType,            // Value field is a 32 bit
                              // integral value
  raatFramedProtocol,         // Value field is a 32 bit
                              // integral value
  raatFramedIPAddress,        // Value field is a 32 bit
                              // integral value
  raatFramedIPNetmask,        // Value field is a 32 bit
                              // integral value
  raatFramedRouting,          // Value field is a 32 bit
                              // integral value
  raatFilterId,               // Value field is a pointer
  raatFramedMTU,              // Value field is a 32 bit
                              // integral value
  raatFramedCompression,      // Value field is a 32 bit
                              // integral value
  raatLoginIPHost,            // Value field is a 32 bit
                              // integral value
  raatLoginService,           // Value field is a 32 bit
                              // integral value
  raatLoginTCPPort,           // Value field is a 32 bit
                              // integral value
  raatUnassigned1,            // Undefined
  raatReplyMessage,           // Value field is a pointer
  raatCallbackNumber,         // Value field is a pointer
  raatCallbackId,             // Value field is a pointer
  raatUnassigned2,            // Undefined
  raatFramedRoute,            // Value field is a pointer
  raatFramedIPXNetwork,       // Value field is a 32 bit
                              // integral value
  raatState,                  // Value field is a pointer
  raatClass,                  // Value field is a pointer
  raatVendorSpecific,         // Value field is a pointer
  raatSessionTimeout,         // Value field is a 32 bit
                              // integral value
  raatIdleTimeout,            // Value field is a 32 bit
```

```
                            // integral value
raatTerminationAction,      // Value field is a 32 bit
                            // integral value
raatCalledStationId,        // Value field is a pointer
raatCallingStationId,       // Value field is a pointer
raatNASIdentifier,          // Value field is a pointer
raatProxyState,             // Value field is a pointer
raatLoginLATService,        // Value field is a pointer
raatLoginLATNode,           // Value field is a pointer
raatLoginLATGroup,          // Value field is a pointer
raatFramedAppleTalkLink,    // Value field is a 32 bit
                            // integral value
raatFramedAppleTalkNetwork,// Value field is a 32 bit
                            // integral value
raatFramedAppleTalkZone,    // Value field is a pointer
raatAcctStatusType,         // Value field is a 32 bit
                            // integral value
raatAcctDelayType,          // Value field is a 32 bit
                            // integral value
raatAcctInputOctets,        // Value field is a 32 bit
                            // integral value
raatAcctOutputOctets,       // Value field is a 32 bit
                            // integral value
raatAcctSessionId,          // Value field is a pointer
raatAcctAuthentic,          // Value field is a 32 bit
                            // integral value
raatAcctSessionTime,        // Value field is a 32 bit
                            // integral value
raatAcctInputPackets,       // Value field is a 32 bit
                            // integral value
raatAcctOutputPackets,      // Value field is a 32 bit
                            // integral value
raatAcctTerminateCause,     // Value field is a 32 bit
                            // integral value
raatAcctMultiSessionId,     // Value field is a pointer
raatAcctLinkCount,          // Value field is a 32 bit
                            // integral value
raatAcctEventTimeStamp = 55,  // Value field is a 32 bit
                                // integral value
raatMD5CHAPChallenge = 60,  // Value field is a pointer
raatNASPortType,            // Value field is a 32 bit
                            // integral value
raatPortLimit,              // Value field is a 32 bit
                            // integral value
```

(continued)

(continued)

```
  raatLoginLATPort,          // Value field is a pointer
  raatARAPPassword = 70      // Value field is a pointer
  raatARAPFeatures,          // Value field is a pointer
  raatARAPZoneAccess,        // Value field is a 32 bit
                             // integral value
  raatARAPSecurity,          // Value field is a 32 bit
                             // integral value
  raatARAPSecurityData,      // Value field is a pointer
  raatPasswordRetry,         // Value field is a 32 bit
                             // integral value
  raatPrompt,                // Value field is a 32 bit
                             // integral value
  raatConnectInfo,           // Value field is a pointer
  raatConfigurationToken,    // Value field is a pointer
  raatEAPMessage,            // Value field is a pointer
  raatSignature,             // Value field is a pointer
  raatAcctInterimInterval = 85,    // Value field is a
                                   // Pointer
  raatARAPChallenge  =  4133,      // Value field is a
                                   // Pointer
  raatARAPGuestLogon,        // Value field is a 32 bit
                             // integral value
  raatARAPChallengeResponse, // Value field is a pointer
  raatReserved = 0xFFFFFFFF  // Undefined
} RAS_AUTH_ATTRIBUTE_TYPE;
```

Values

raatMinimum

Specifies a value that is equal to zero, and used as the null-terminator in any array of **RAS_AUTH_ATTRIBUTE** structures.

raatUserName

Specifies the name of the user to be authenticated. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatUserPassword

Specifies the password of the user to be authenticated. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatMD5CHAPPassword

Specifies the password provided by the user in response to an MD5 Challenge Handshake Authentication Protocol (CHAP) challenge. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatNASIPAddress

Specifies the Network Access Server (NAS) IP address. An Access-Request should specify either an NAS IP address or an NAS identifier. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatNASPort

Specifies the physical or virtual private network (VPN) through which the user is connecting to the NAS. Note that this value is not a port number in the sense of TCP or UDP. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatServiceType

Specifies the type of service the user has requested or the type of service to be provided. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatFramedProtocol

Specifies the type of framed protocol to use for framed access, for example SLIP, PPP, or ARAP (AppleTalk Remote Access Protocol). The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatFramedIPAddress

Specifies the IP address to be configured for the user requesting authentication. This attribute is typically returned by the authentication provider. However, the NAS may use it in an authentication request to specify a preferred IP address. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatFramedIPNetmask

Specifies the IP network mask for a user that is a router to a network. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatFramedRouting

Specifies the routing method for a user that is a router to a network. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatFilterId

Specifies the filter list for the user requesting authentication. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatFramedMTU

Specifies the Maximum Transmission Unit (MTU) for the user. This attribute is used in cases where the MTU is not negotiated through some other means, such as PPP. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatFramedCompression
Specifies a compression protocol to use for the connection. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatLoginIPHost
Specifies the system with which to connect the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatLoginService
Specifies the service to use to connect the user to the host specified by *raatLoginIPHost*. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatLoginTCPPort
Specifies the port to which to connect the user. This attribute is present only if the *raatLoginService* attribute is present. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatUnassigned1
This value is currently unassigned.

raatReplyMessage
Specifies a message to display to the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatCallbackNumber
Specifies a callback number. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatCallbackId
Specifies a location to call back. The value of this attribute is interpreted by the NAS. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatUnassigned2
This value is currently unassigned. The value field in **RAS_AUTH_ATTRIBUTE** for this type is also undefined.

raatFramedRoute
Specifies routing information to configure on the NAS for the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatFramedIPXNetwork
Specifies the IPX network number to configure for the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatState
Refer to *RFC 2138* for detailed information about this value. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer.

raatClass
Specifies a value that is provided to the NAS by the authentication provider. The NAS should use this value when communicating with the accounting provider. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatVendorSpecific
Specifies a field for extended attributes. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatSessionTimeout
Specifies the maximum number of seconds for which to provide service to the user. After this time, the session is terminated. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatIdleTimeout
Specifies the maximum number of consecutive seconds the session can be idle. If the idle time exceeds this value, the session is terminated. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatTerminationAction
Refer to the above-referenced files at ds.internic.net for detailed information about this value. The value field in **RAS_AUTH_ATTRIBUTE** for this type is 32-bit integral value. For more information, see *RFC 2138*.

raatCalledStationId
Specifies the number that the user dialed to connect to the NAS. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatCallingStationId
Specifies the number from which the user is calling. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatNASIdentifier
Specifies the NAS identifier. An Access-Request should specify either an NAS identifier or an NAS IP address. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatProxyState
Specifies a value that a proxy server includes when forwarding an authentication request. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatLoginLATService
Specifies an attribute that is not currently used for authentication on Windows 2000. For more information, see *RFC 2138*.

raatLoginLATNode
Specifies an attribute that is not currently used for authentication on Windows 2000. For more information, see *RFC 2138*.

raatLoginLATGroup
Specifies an attribute that is not currently used for authentication on Windows 2000. For more information, see *RFC 2138*.

raatFramedAppleTalkLink
Specifies the AppleTalk network number for the user when the user is another router. The value field in **RAS_AUTH_ATTRIBUTE** for this type is 32-bit integral value. For more information, see *RFC 2138*.

raatFramedAppleTalkNetwork
Specifies the AppleTalk network number that the NAS should use to allocate an AppleTalk node for the user. This attribute is used only when the user is not another router. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatFramedAppleTalkZone
Specifies the AppleTalk default zone for the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatAcctStatusType
Specifies whether the accounting provider should start or stop accounting for the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctDelayType
Specifies the length of time that the client has been attempting to send the current request. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctInputOctets
Specifies the number of octets that have been received during the current accounting session. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctOutputOctets
Specifies the number of octets that were sent during the current accounting session. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctSessionId
Specifies a value to enable the identification of matching start and stop records within a log file. The start and stop records are sent in the *raatAcctStatusType* attribute. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2139*.

raatAcctAuthentic
Specifies, to the accounting provider, how the user was authenticated; for example by Windows 2000 Directory Services, RADIUS, or some other authentication provider. The value field in **RAS_AUTH_ATTRIBUTE**for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctSessionTime
Specifies the number of seconds that have elapsed in the current accounting session. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctInputPackets
Specifies the number of packets that have been received during the current accounting session. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctOutputPackets
Specifies the number of packets that have been sent during the current accounting session. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctTerminateCause
Specifies how the current accounting session was terminated. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctMultiSessionId
Specifies a value to enable the identification of related accounting sessions within a log file. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctLinkCount
Specifies the number of links if the current accounting session is using a multilink connection. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2139*.

raatAcctEventTimeStamp
Specifies an attribute that is included in an accounting request packet. It specifies the time that the event took place. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see the *Radius Extensions 04* internet draft.

raatMD5CHAPChallenge
Specifies the CHAP challenge sent by the NAS to a CHAP user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see *RFC 2138*.

raatNASPortType
Specifies the type of the port through which the user is connecting, for example, asynchronous, ISDN, virtual. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatPortLimit
Specifies the number of ports the NAS should make available to the user for multilink sessions. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see *RFC 2138*.

raatLoginLATPort
Specifies an attribute that is not currently used for authentication on Windows 2000. Please refer to the above-referenced files at ds.internic.net for detailed information about this value.

raatARAPPassword
Specifies a password to use for AppleTalk Remote Access Protocol (ARAP) authentication. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatARAPFeatures
Specifies information that an NAS should send back to the user in an ARAP "feature flags" packet. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatARAPZoneAccess
Specifies how to use the ARAP zone list for the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatARAPSecurity
Specifies an ARAP security module to use during a secondary authentication phase between the NAS and the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatARAPSecurityData
Specifies the data to use with an ARAP security module. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatPasswordRetry
Specifies the number of password retry attempts to permit the user. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value.

raatPrompt
Specifies whether the NAS should echo the user response to a challenge. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatConnectInfo
Specifies information about the type of connection the user is using. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a Pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatConfigurationToken
Specifies user-profile information in communications between RADIUS Proxy Servers and RADIUS Proxy Clients. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatEAPMessage
Specifies that EAP information be sent directly between the user and the authentication provider. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatSignature
Specifies a signature to include with CHAP, EAP, or ARAP packets. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatAcctInterimInterval
Specifies the time, in seconds, between accounting updates. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatARAPChallenge
Specifies a Apple Remote Access Protocol (ARAP) challenge. In ARAP, both the server and the client may issue challenges. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatARAPGuestLogon
Specifies a Apple Remote Access Protocol (ARAP) guest logon. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a 32-bit integral value. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatARAPChallengeResponse
Specifies the response to a Apple Remote Access Protocol (ARAP) challenge. In ARAP, either the server or the client may respond to challenges. The value field in **RAS_AUTH_ATTRIBUTE** for this type is a pointer. For more information, see the Internet draft, *draft-ietf-radius-ext-04*.

raatReserved
The value field in **RAS_AUTH_ATTRIBUTE** for this type is undefined.

! Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Raseapif.h.

+ See Also

Extensible Authentication Protocol Reference, EAP Enumerated Types, **RAS_AUTH_ATTRIBUTE**

Extensible Authentication Protocol COM Interfaces

Implement the following COM interfaces when implementing an authentication protocol for Microsoft® Windows® 2000 Server:

IEAPProviderConfig

IEAPProviderConfig

When to Implement

Implement the **IEAPProviderConfig** interface to provide a configuration UI for an EAP provider. This interface is for configuring the EAP provider on the server. For information about the client-side configuration, see the reference page for the **RasEapInvokeConfigUI** function.

When to Use

The system calls the methods of this interface when a user chooses to configure an EAP provider in the R RAS snap-in.

Methods in Vtable Order

IUnknown Methods	Description
Query Interface	Returns pointers to supported interfaces
AddRef	Increments reference count
Release	Decrements reference count

IEAPProviderConfig Methods	Description
Initialize	Initializes an EAP configuration session
Uninitialize	Shuts down an EAP configuration session
ServerInvokeConfigUI	Invokes the EAP configuration user interface
RouterInvokeConfigUI	
RouterInvokeCredentialsUI	

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Rrascfg.h.

See Also

Extensible Authentication Protocol Reference, Extensible Authentication Protocol COM Interfaces

IEAPProviderConfig::Initialize

The system calls the **IEAPProviderConfig::Initialize** method to initialize an EAP configuration session with the specified computer.

```
HRESULT Initialize(
  LPCOLESTR pszMachineName,      // pointer to computername
  DWORD dwEapTypeId              // specifies the EAP
  ULONG_PTR * puConnectionParam  // pointer to config
                                 // session ID
);
```

Parameters

pszMachineName
Pointer to a string that contains the name of the computer on which to configure EAP.

dwEapTypeId
Specifies the EAP for which to initialize a configuration session.

puConnectionParam
Pointer to an unsigned integer variable. On successful return, the value of this variable identifies this configuration session.

Return Values

If the function succeeds, the return value should be S_OK.

If the function fails, the return value should be one of the following codes.

Value	Description
E_FAIL	Non-specific error
E_INVALIDARG	One of the arguments is invalid
E_OUTOFMEMORY	The method failed because it was unable to allocate required memory
E_UNEXPECTED	An unexpected error occurred

Remarks

The configuration UI should allow the user to configure the EAP provider on a remote computer. Establish the connection to the remote computer during the call to **IEAPProviderConfig::Initialize**.

The DLL that implements **IEAPProviderConfig** may support more than one authentication protocol. The *dwEapTypeId* parameter specifies for which authentication protocol to initialize a configuration session.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Rrascfg.h.

See Also

Extensible Authentication Protocol Reference, Extensible Authentication Protocol COM Interfaces, **IEAPProviderConfig**, **IEAPProviderConfig::RouterInvokeConfigUI**, **IEAPProviderConfig::RouterInvokeCredentialsUI**, **IEAPProviderConfig::ServerInvokeConfigUI**, **IEAPProviderConfig::Uninitialize**

IEAPProviderConfig::Uninitialize

The system calls the **IEAPProviderConfig::Uninitialize** method to shutdown the specified EAP configuration session.

```
HRESULT Uninitialize(
  DWORD dwEapTypeId          // specifies the EAP
  ULONG_PTR uConnectionParam  // config session ID
);
```

Parameters

dwEapTypeId
Specifies the EAP for which to shut down the configuration session.

uConnectionParam
Specifies the configuration session to shut down.

Return Values

If the function succeeds, the return value should be S_OK.

If the function fails, the return value should be one of the following codes.

Value	Description
E_FAIL	Non-specific error
E_INVALIDARG	One of the arguments is invalid
E_OUTOFMEMORY	The method failed because it was unable to allocate required memory
E_UNEXPECTED	An unexpected error occurred

Remarks

The configuration UI should allow the user to configure the EAP provider on a remote computer. Delete the connection to the remote computer during the call to **IEAPProviderConfig::Uninitialize**.

The DLL that implements **IEAPProviderConfig** may support more than one authentication protocol. The *dwEapTypeId* parameter specifies for which authentication protocol to shut down the configuration session.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Rrascfg.h.

See Also

Extensible Authentication Protocol Reference, Extensible Authentication Protocol COM Interfaces, **IEAPProviderConfig**, **IEAPProviderConfig::Initialize**, **IEAPProviderConfig::RouterInvokeConfigUI**, **IEAPProviderConfig::RouterInvokeCredentialsUI**, **IEAPProviderConfig::ServerInvokeConfigUI**

IEAPProviderConfig::ServerInvokeConfigUI

The system calls the **IEAPProviderConfig::ServerInvokeConfigUI** method to invoke the configuration user interface for EAP authentication between a remote access client and server.

```
HRESULT Configure(
  DWORD dwEapTypeId             // specifies the EAP
  ULONG_PTR uConnectionParam,   // config session id
  HWND hWnd,                    // handle to parent window
  ULONG_PTR dwReserved1,        // reserved, should be zero
  ULONG_PTR dwReserved2         // reserved, should be zero
);
```

Parameters

dwEapTypeId
Specifies the EAP for which to invoke the configuration user interface.

uConnectionParam
Specifies the configuration session for which to invoke the user interface.

hWnd
Handle to the parent window for the configuration user interface.

dwReserved1
This parameter is reserved and should be zero.

dwReserved2
This parameter is reserved and should be zero.

Return Values

If the function succeeds, the return value should be S_OK.

If the function fails, the return value should be one of the following codes.

Value	Description
E_FAIL	Non-specific error
E_INVALIDARG	One of the arguments is invalid
E_OUTOFMEMORY	The method failed because it was unable to allocate required memory
E_UNEXPECTED	An unexpected error occurred

Remarks

The DLL that implements **IEAPProviderConfig** may support more than one authentication protocol. The *dwEapTypeId* parameter specifies for which authentication protocol to invoke the configuration user interface.

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Rrascfg.h.

See Also

Extensible Authentication Protocol Reference, Extensible Authentication Protocol COM Interfaces, **IEAPProviderConfig**, **IEAPProviderConfig::Initialize**, **IEAPProviderConfig::Uninitialize**, **IEAPProviderConfig::RouterInvokeConfigUI**, **IEAPProviderConfig::RouterInvokeCredentialsUI**

IEAPProviderConfig::RouterInvokeConfigUI

The system calls the **IEAPProviderConfig::RouterInvokeConfigUI** method to invoke the configuration user interface for EAP authentication between two routers:

```
HRESULT RouterInvokeConfigUI (
  DWORD dwEapTypeId               // specifies the EAP
  ULONG_PTR uConnectionParam,     // config session id
  HWND hwndParent,                // handle to parent window
  DWORD dwFlags,                  // flag specifying router
                                  // to router authentication
  BYTE * pConnectionDataIn,       // current config data
  DWORD dwSizeOfConnectionDataIn,   // size of current
                                    // config data
  BYTE ** ppConnectionDataOut,        // new config data
  DWORD * pdwSizeOfConnectionDataOut  // size of new config
                                      // data
);
```

Parameters

dwEapTypeId
Specifies the EAP for which to invoke the configuration user interface.

uConnectionParam
Specifies the configuration session for which to invoke the user interface.

hwndParent
Handle to the parent window for the configuration user interface.

dwFlags
Specifies the RAS_EAP_FLAG_ROUTER flag. This is the only valid flag for this parameter and it indicates that authentication is between two routers. This parameter will always include this flag.

pConnectionDataIn
Pointer to the current configuration data for the interface.

dwSizeOfConnectionDataIn
Specifies the size of the current configuration data pointed to by the *pConnectionDataIn* parameter.

ppConnectionDataOut
Pointer to a pointer to a buffer that contains the new configuration data for the interface.

pdwSizeOfConnectionDataOut
Pointer to a DWORD variable to receive the size of the new configuration data.

Return Values

If the function succeeds, the return value should be S_OK.

If the function fails, the return value should be one of the following codes.

Value	Description
E_FAIL	Non-specific error
E_INVALIDARG	One of the arguments is invalid
E_OUTOFMEMORY	The method failed because it was unable to allocate required memory
E_UNEXPECTED	An unexpected error occurred

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Rrascfg.h.

See Also

Extensible Authentication Protocol Reference, Extensible Authentication Protocol COM Interfaces, **IEAPProviderConfig**, **IEAPProviderConfig::Initialize**, **IEAPProviderConfig::RouterInvokeCredentialsUI**, **IEAPProviderConfig::ServerInvokeConfigUI**, **IEAPProviderConfig::Uninitialize**

IEAPProviderConfig::RouterInvokeCredentialsUI

The system calls the **IEAPProviderConfig::RouterInvokeCredentialsUI** method to invoke the credentials user interface for EAP authentication between two routers.

```
HRESULT RouterInvokeCredentialsUI (
  DWORD dwEapTypeId               // specifies the EAP
  ULONG_PTR uConnectionParam,     // config session id
  HWND hwndParent,                // handle to parent window
  DWORD dwFlags,                  // flag specifying router
                                  // to router authentication
  BYTE* pConnectionDataIn,            // current config data
  DWORD dwSizeOfConnectionDataIn,     // size of current
                                      // config data
  BYTE* pUserDataIn,              // current credentials
  DWORD dwSizeOfUserDataIn,       // size of current
                                  // credentials
  BYTE ** ppUserDataOut,          // new credentials
  DWORD * pdwSizeOfUserDataOut    // size of new credentials
);
```

Parameters

dwEapTypeId
: Specifies the EAP for which to invoke the configuration user interface.

uConnectionParam
: Specifies the configuration session for which to invoke the user interface.

hwndParent
: Handle to the parent window for the configuration user interface.

dwFlags
: Specifies the RAS_EAP_FLAG_ROUTER flag. This is the only valid flag for this parameter and it indicates that authentication is between two routers. This parameter will always include this flag.

pConnectionDataIn
: Pointer to the current configuration data for the interface.

dwSizeOfConnectionDataIn
: Specifies the size of the current configuration data pointed to by the *pConnectionDataIn* parameter.

pUserDataIn
Pointer to the current credential data for the interface.

dwSizeOfUserDataIn
Specifies the size of the current credentials data.

ppUserDataOut
Pointer to a pointer to a buffer to receive the new credentials data for the interface.

pdwSizeOfUserDataOut
Pointer to a **DWORD** variable to receive the size of the new credentials data.

Return Values

If the function succeeds, the return value should be S_OK.

If the function fails, the return value should be one of the following codes.

Value	Description
E_FAIL	Non-specific error
E_INVALIDARG	One of the arguments is invalid
E_OUTOFMEMORY	The method failed because it was unable to allocate the required memory
E_UNEXPECTED	An unexpected error occurred

Requirements

Windows NT/2000: Requires Windows 2000.
Header: Declared in Rrascfg.h.

See Also

Extensible Authentication Protocol Reference, Extensible Authentication Protocol COM Interfaces, **IEAPProviderConfig**, **IEAPProviderConfig::Initialize**, **IEAPProviderConfig::RouterInvokeConfigUI**, **IEAPProviderConfig::ServerInvokeConfigUI**, **IEAPProviderConfig::Uninitialize**

CHAPTER 14

Tracing

Tracing Overview

The following documentation describes the implementation of the common tracing DLL, which provides a uniform mechanism for generating diagnostic output for the Microsoft® Windows NT®/Windows® 2000 Routing and RAS components as well as any other application that wishes to use the DLL. The DLL provides dynamic configuration change, allowing a user to direct output to a console or to a specified file. In the case of files, the user can specify the maximum size for the file.

Using Tracing

Each application or service component calls **TraceRegister** to obtain an Identifier (ID) to use in calls to the output functions. On this call, the DLL reads configuration information for the caller from the registry, and sets up the console or file to which output will be sent. In addition, a critical section is created that will be used to synchronize calls to the tracing DLL functions by the registering component's threads. An event is associated with the registry key for the caller, so that changes to the tracing parameters for the caller can be handled dynamically.

After registering, the application may call the output functions, passing the ID returned by **TraceRegister**. When the application no longer requires the tracing DLL's support, it should call **TraceDeregister** so that handles associated with it can be closed.

There are two versions of each output function. One version prefixes the output it generates with standard information such as the name associated with the caller, the thread ID of the caller, and the current time. The other version allows the caller to omit the standard information normally generated. For instance, **TracePrintf** includes standard information, but **TracePrintfEx** does not, unless the flag passed to it specifies that it should.

Support for Unicode clients is built into the tracing DLL. All that is required is that the client define the constant UNICODE before including the header containing definitions for the tracing DLL functions.

Configuration

In order to enable console tracing, the value **EnableConsoleTracing** must exist under the registry key

HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\TRACING

and be non-zero. If this value does not exist, or is zero, console tracing is disabled. This value is read when rtutils.dll is loaded; changes to this value after rtutils.dll is already loaded will have no effect until the DLL is unloaded and loaded again.

In addition to the preceding "global" value, the registry may also contain values for individual clients. When a client "xyz" calls **TraceRegister**, the tracing DLL looks under the registry key

HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\TRACING\XYZ

for the following values:

- **EnableConsoleTracing**: this is a **REG_DWORD** that defaults to zero; tracing to the console is enabled if this value is non-zero.
- **EnableFileTracing**: this is a **REG_DWORD** that defaults to zero; tracing to a file named XYZ.LOG is enabled if this value is non-zero.
- **ConsoleTracingMask**: this is a **REG_DWORD** that defaults to 0xFFFF0000; the bits in the high-order word correspond to components in the client. If a call to one of the extended output functions has the flag **TRACE_USE_MASK** set, this registry value is compared against the high-order word of the flag passed to the output function, to decide whether or not to send the output to the console.
- **FileTracingMask**: this is a **REG_DWORD** that defaults to 0xFFFF0000; it operates similarly to **ConsoleTracingMask**.
- **MaxFileSize:** this is a **REG_DWORD** that defaults to 0x10000; this is the maximum size a tracing file can grow to before it is renamed.
- **FileDirectory**: this is a **REG_EXPAND_SZ** that defaults to %WINDIR%\TRACING; this is the directory in which the tracing file is created.

The defaults are used only if the key is found (or can be created) but some values are absent from the key. If the registry key is not found and cannot be created, the call to **TraceRegister** fails.

Alternatively, a client "xyz" could call **TraceRegisterEx**, which takes a flag allowing the caller to specify the settings to use. Thus, a client could use the tracing DLL without creating any key in the registry. For instance:

```
TraceRegisterEx("xyz", TRACE_USE_CONSOLE);
```

would register the client "xyz" to use the console for tracing, and the tracing DLL would not attempt to read the registry key for the client. Similarly.

```
TraceRegisterEx("abc", TRACE_USE_FILE);
```

would register the client "abc" to use a file for tracing, bypassing the registry key for the client. However, for console tracing, using **TraceRegisterEx** still requires that the global **EnableConsoleTracing** value exist under

HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\TRACING

and be non-zero.

Console Manipulation

The tracing DLL creates a thread that runs in the background, detecting changes to the configuration of clients that use the registry, as well as handling the following key-presses in the console:

Key pressed	Action taken
Control-Tab	displays the screen for the next console client
Control-Shift-Tab	displays the screen for the previous console client
Pause	toggles tracing for the displayed console client
Space-bar	toggles tracing for the displayed console client
Up-arrow	moves screen up by one line
Down-arrow	moves screen down by one line
Left-arrow	moves screen left by one column
Right-arrow	moves screen right by one column
Page-up	moves screen up by one page
Page-down	moves screen down by one page

Tracing Reference

Use the following functions to add tracing functionality to your software:

TraceDeregister
TraceDump
TraceDumpEx
TracePrintf
TracePrintfEx
TracePuts
TracePutsEx
TraceRegister
TraceRegisterEx
TraceVprintf
TraceVprintfEx

TraceDeregister

The **TraceDeregister** function frees resources and closes files associated with tracing registration on behalf the calling service or application. Call TraceDeregister no more than once for a service or application, regardless of how many calls were made on the service or application's behalf.

```
DWORD TraceDeregister(
  IN DWORD dwTraceID                // handle from initial
                                    // TraceRegister call
);
```

Parameters

dwTraceID
The handle returned by the calling service or application's initial **TraceRegister** call.

Return Values

If the function succeeds, the return value is 0.

If the function fails, the return value is an error code. Call **GetLastError** for further information.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

See Also

About Tracing, Tracing Reference, **TraceRegister**, **TracePrintf**, **TraceVprintf**, **TracePuts**, **TraceDump**

TraceDump

The **TraceDump** function outputs a hexadecimal dump of size *dwByteCount*, prefixed with the name associated with the calling service or application's *dwTraceID*, the associated Thread identifier used with the RRAS tracing functionality, the current system time, and a brief description of the dump.

```
DWORD TraceDump(
  IN DWORD dwTraceID,         // handle from initial
                              // TraceRegister call
```

```
  IN LPBYTE lpbBytes,          // pointer to dump buffer
  IN DWORD dwByteCount,        // number of bytes to dump
  IN DWORD dwGroupSize,        // size of byte grouping
                               // (1,2 or4)
  IN BOOL bAddressPrefix,      // include memory address
                               // toggle
  IN LPCTSTR lpszPrefix        // prefix
);
```

Parameters

dwTraceID
The handle returned by the calling service or application's initial **TraceRegister** call.

lpbBytes
A pointer to the buffer from which the hex dump is to be generated

dwByteCount
The number of bytes to dump from the buffer.

dwGroupSize
The output's byte grouping size. Valid values are 1, 2, or 4.

bAddressPrefix
Boolean value that determines whether each line of the hex dump has its memory address as a prefix. A value of **TRUE** includes the memory address.

lpszPrefix
Pointer to the prefix.

Return Values

Successful execution of **TraceDump** returns the number of characters output.

Otherwise, **TraceDump** returns zero. Call **GetLastError** to get the error code.

Remarks

TraceDump generates debug style dumps, with the byte-ordering dependent on the processor's endian setting. Also note that the last line of the dump is padded with zeroes.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

See Also

About Tracing, Tracing Reference, **TraceDumpEx**, **TraceRegister**, **TraceDeregister**, **TracePrintf**, **TraceVprintf**, **TracePuts**

TraceDumpEx

The **TraceDumpEx** function outputs a hexadecimal dump of size *dwByteCount*. **TraceDumpEx** differentiates itself from **TraceDump** in its Extended (Ex) output options, implemented through the use of non-zero *dwFlags* values. Output from **TraceDumpEx** can include information with a prefix of the name associated with the calling service or application's *dwTraceID*, the associated Thread identifier used with the RRAS tracing functionality, the current system time, and a brief description of the dump.

```
DWORD TraceDumpEx(
  IN DWORD dwTraceID,
  IN DWORD dwFlags,     // OPTIONAL
  IN LPBYTE lpbBytes,
  IN DWORD dwByteCount,
  IN DWORD dwGroupSize,
  IN BOOL bAddressPrefix,
  IN LPCTSTR lpszPrefix
);
```

Parameters

dwTraceID
The handle returned by the calling service or application's initial **TraceRegister** call.

dwFlags
Flags that control appearance of **TraceDumpEx** output. Ensure *dwFlags* is one or more of the following:

TRACE_NO_STDINFO
Suppresses output of the standard information associated with *dwTraceID.*

TRACE_USE_MASK
Determines whether file and/or console output will be generated by comparing the high-order word of *dwFlags* against registry values **FileTracingMask** and **ConsoleTracingMask**.

lpbBytes
A pointer to the buffer from which the hex dump is to be generated.

dwByteCount
The number of bytes to dump from the buffer.

dwGroupSize
The output's byte grouping size. Valid values are 1, 2, or 4.

bAddressPrefix
Boolean value that determines whether each line of the hex dump is prefixed with its memory address. A value of **TRUE** includes the memory address.

lpszPrefix
Pointer to the prefix.

Return Values

Successful execution of **TraceDump** returns the number of characters output.

Otherwise, **TraceDump** returns zero. Call **GetLastError** to get the error code.

Remarks

TraceDumpEx generates debug style dumps, with the byte-ordering dependent on the processor's endian setting. Also note that the last line of the dump is padded with zeroes.

! Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

+ See Also

About Tracing, Tracing Reference, **TraceDump**, **TraceRegisterEx**, **TracePrintfEx**, **TraceVprintfEx**, **TracePutsEx**

TracePrintf

The **TracePrintf** function outputs tracing information, including the following: calling service or application's name, the current time, and tracing information in the format specified by the optional argument or arguments included in *lpszFormat*. See the note below for an example of **TracePrintf** results.

```
DWORD TracePrintf(
  DWORD dwTraceID,          // handle from initial
                            // TraceRegister call
  LPCTSTR lpszFormat,       // printf()-style formatting
                            // information
  ...                       // one or more optional arguments
);
```

Parameters

dwTraceID
The handle returned by the calling service or application's initial **TraceRegister** call.

lpszFormat
Pointer to a null-terminated string containing **printf**-style format control information.

...
One or more optional arguments, depending on the format control specified in *lpszFormat*.

Return Values

If the function succeeds, **TracePrintf** returns the number of characters output, excluding the terminating null-character.

If the function fails, the return value will be zero. This may also indicate that tracing is disabled in the registry. See *Tracing Configuration* for more information.

Remarks

The following is an example for the output from **TracePrintf**. In the following example, the service or application calling **TracePrintf** is IPRIP, and its associated Thread identifier for use with the RRAS tracing functionality is 129:

```
[IPRIP:129] 21:01:20: new entry: dest=0.0.0.0, nexthop=157.55.80.1, metric=1,
protocol=2
[IPRIP:129] 21:01:20: received RIP v1 response from 157.55.84.244 on address
157.55.94.40
```

To suppress the prefixes, use **TracePrintfEx**.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

See Also

About Tracing, Tracing Reference, **TracePrintfEx**, **TraceRegister**, **TraceDeregister**, **TraceVprintf**, **TracePuts**, **TraceDump**

TracePrintfEx

The **TracePrintfEx** function outputs tracing information. **TracePrintfEx** differentiates itself from **TracePrintf** by offering Extended options (Ex) implemented through non-zero *dwFlags* values. Output generated by **TracePrintfEx** includes up to the following: calling service or application name, the current time, and tracing information in the format specified by the optional included in *lpszFormat*.

If *dwFlags* is zero, **TracePrintfEx** behaves exactly as **TracePrintf**.

```
DWORD TracePrintfEx(
  DWORD dwTraceID,        // handle returned by TraceRegister
  DWORD dwFlags,          // flags to control output
  LPCTSTR lpszFormat,     // pointer to printf-style
```

```
                        // format string
   ...                  // optional args, which depend on
                        // format string.
);
```

Parameters

dwTraceID

The handle returned by the calling service or application's initial **TraceRegister** call.

dwFlags

Specifies optional flags that control appearance of **TracePrintfEx** output. Ensure *dwFlags* is one or more of the following:

TRACE_NO_STDINFO

Suppresses output of the standard information associated with *dwTraceID.*

TRACE_USE_MASK

Determines whether file and/or console output will be generated by comparing the high-order word of *dwFlags* against registry values **FileTracingMask** and **ConsoleTracingMask.**

lpszFormat

Pointer to a null-terminated string containing **printf**-style format control information.

...

One or more optional arguments, depending on the format control specified in *lpszFormat*.

Return Values

If the function is successful, **TracePrintfEx** returns the number of characters output, excluding the terminating null-character.

If the function fails, the return value is zero. This may indicate that tracing is disabled in the registry. See *Tracing Configuration* for more information.

! Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

+ See Also

About Tracing, Tracing Reference, **TracePrintf**, **TraceRegisterEx**, **TraceVprintfEx**, **TracePutsEx**, **TraceDumpEx**

TracePuts

The **TracePuts** function is an efficient way to retrieve information associated with a registered service or application's identifier (*dwTraceID*). **TracePuts** also outputs the string literal passed as the function's second argument.

```
DWORD TracePuts(
  IN DWORD dwTraceID,
  IN LPCTSTR lpszString
);
```

Parameters

dwTraceID
The handle returned by the calling service or application's initial **TraceRegister** call.

lpszString
The string to be output.

Return Value

If the function is successful, **TracePuts** returns the number of characters output, excluding the terminating null-character.

If the function fails, the return value is zero. This may also indicate that tracing is disabled in the registry. See *Tracing Configuration* for more information.

Remarks

TracePuts outputs the name associated with *dwTraceID*, the internal thread identifier used to identify the caller, the current time, and the literal string specified by *lpszString*. Since TracePuts performs no formatting on its arguments, it is more efficient than **TracePrintf** or **TraceVprintf**. To suppress the prefixes and prevent output from starting on a new line, see ***TracePutsEx***.

! Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

+ See Also

About Tracing, Tracing Reference, **TracePutsEx**, **TraceRegister**, **TraceDeregister**, **TracePrintf**, **TraceVprintf**, **TraceDump**

TracePutsEx

The **TracePutsEx** function is an efficient way to retrieve information associated with a registered service or application's identifier (*dwTraceID*). **TracePutsEx** differs from **TracePuts** in its Extended (Ex) flexibility with regard to output, achieved through the use of non-zero flags implemented with *dwFlags*. **TracePutsEx** also outputs the string literal passed as the function's second argument.

```
DWORD TracePutsEx(
  IN DWORD dwTraceID,
  IN DWORD dwFlags,     // OPTIONAL
  IN LPCTSTR lpszString
);
```

Parameters

dwTraceID
The handle returned by the calling service or application's initial **TraceRegister** call.

dwFlags
Flags that control appearance of **TracePutsEx** output. Ensure *dwFlags* is one or more of the following:

TRACE_NO_STDINFO
Suppresses output of the standard information associated with *dwTraceID.*

TRACE_USE_MASK
Determines whether file and/or console output will be generated by comparing the high-order word of *dwFlags* against registry values **FileTracingMask** and **ConsoleTracingMask**.

lpszString
The string to be output.

Return Values

If the function is successful, **TracePuts** returns the number of characters output, excluding the terminating null-character.

If the function fails, the return value is zero. This may indicate that tracing is disabled in the registry. See *Tracing Configuration* for more information.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

See Also

About Tracing, Tracing Reference, **TracePuts**, **TraceRegisterEx**, **TracePrintfEx**, **TraceVprintfEx**, **TraceDumpEx**

TraceRegister

Use the **TraceRegister** function to register services or applications with the tracing DLL. Its successful return value is an identifier that provides a handle to subsequent tracing functions available in Windows NT/Windows 2000. This function, or its extended functionality counterpart **TraceRegisterEx**, must be called before any other tracing functions are called. **TraceDeregister** or **TraceDeregisterEx** should be called when trace functions are no longer needed, in order to free resources.

```
DWORD TraceRegister(
  IN LPCTSTR lpszCallerName    // caller name
);
```

Parameters

lpszCallerName

Pointer to a null-terminated string containing the service or application name being registered. This is the name with which the service tracing functions will identify the caller.

Return Values

If successful, this function will return a **DWORD** to be used as the service or application's identifier (handle) for subsequent calls to tracing functions.

If the function fails, INVALID_TRACEID is returned. This indicates the caller could not be registered. Call **GetLastError** to retrieve the error code.

Remarks

Upon successful execution of **TraceRegister**, configuration for the service or application calling **TraceRegister** will be created and kept in the registry path \System\CurrentControlSet\Services\Tracing\<*lpszCallerName*> under the HKEY_LOCAL_MACHINE key. Such configuration parameters are kept intact, even if the service or application is deregistered from tracing utilities by calling TraceDeregister. If the registry entries cannot be created, the call to **TraceRegister** will fail. There are certain values within this key that can be modified to change the behavior of trace output.

EnableConsoleTracing

A REG_DWORD that determines whether tracing to the console is enabled. Console tracing is enabled if the value is non-zero. The default value is 1.

EnableFileTracing
A REG_DWORD that determines whether tracing information should be sent to a file called *lpzsCallerName*.log. File tracing is enabled if the value is non-zero. The default value is 1.

ConsoleTracingMask
A REG_DWORD that regulates whether output from an extended tracing function call is directed to the console. The bits in the high-order word correspond to components in the client; if a call to an extended output function has the flag TRACE_USE_MASK set, the value of ConsoleTracingMask is compared to the flag sent to the function to determine whether to send output to the console. The default value is 0xFFFF0000.

FileTracingMask
A REG_DWORD that works in a similar way to ConsoleTracingMask, regulating whether the extended tracing function calls direct their output to File Tracing. The default value is 0xFFFF0000.

MaxFileSize
A REG_DWORD that defines the maximum size a tracing file can become before it is renamed. The default value is 0x10000.

FileDirectory
A REG_EXPAND_SZ that controls the directory in which the tracing file is created. The default is %windir%\tracing.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

See Also

About Tracing, Tracing Reference, **TraceRegisterEx**, **TraceDeregister**, **TracePrintf**, **TraceVprintf**, **TracePuts**, **TraceDump**

TraceRegisterEx

The **TraceRegisterEx** function registers services or applications with the tracing DLL. **TraceRegisterEx** differentiates itself from **TraceRegister** by providing Extended flexibility (Ex) with regard to the creation or reading of registry keys.

Successful execution of **TraceRegisterEx** returns an identifier used as a handle to subsequent tracing functions available in Microsoft® Windows NT®/Windows® 2000. This function, or its counterpart **TraceRegister**, must be called before any other tracing functions are called. If no flags are passed to **TraceRegisterEx** (if *dwFlags* is zero), **TraceRegisterEx** behaves exactly as **TraceRegister**. **TraceDeregister** or **TraceDeregisterEx** should be called when trace functions are no longer needed, in order to free resources.

```
DWORD TraceRegisterEx(
  IN LPCTSTR lpszCallerName,
  IN DWORD dwFlags             // OPTIONAL
);
```

Parameters

lpszCallerName

A pointer to a null-terminated string containing the service or application name being registered. This is the name with which the service tracing functions will identify the caller.

dwFlags

Flags that control the nature of the calling service or application's registration. Ensure *dwFlags* is one or more of the following:

TRACE_USE_CONSOLE

Tracing output is sent to the console; using this parameter avoids loading or writing settings from the registry.

TRACE_USE_FILE

Tracing output is sent to a file; using this parameter avoids reading or writing settings from the registry.

Return Values

Success will return a **DWORD** to be used as the service or application identifier (handle) for subsequent calls to tracing functions.

INVALID_TRACEID

The caller could not be registered. Call **GetLastError** to retrieve the error code.

Remarks

Upon successful execution of **TraceRegisterEx**, configuration for the service or application calling **TraceRegisterEx** will be taken from the parameters passed as *dwFlags*, and registry reads or writes will not occur. If the value of *dwFlags* is zero, then a call to **TraceRegisterEx** will behave exactly as a call to **TraceRegister**, and initialization and configuration parameters will be created and kept in the registry path \System\CurrentControlSet\Services\Tracing\<*lpszCallerName*> under the HKEY_LOCAL_MACHINE key.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

See Also

About Tracing, Tracing Reference, **TraceRegister**, **TracePrintfEx**, **TraceVprintfEx**, **TracePutsEx**, **TraceDumpEx**

TraceVprintf

Functionality of **TraceVprintf** is very similar to that of **TracePrintf**, except that it takes a prepared variable argument list as its third variable. See ***TracePrintf*** for more information.

```
DWORD TraceVprintf(
  IN DWORD dwTraceID,       // handle from initial
                            // TraceRegister call
  IN LPCTSTR lpszFormat,    // printf-style formatting
                            // information
  IN va_list arglist        // prepared list of variable
                            // arguments
);
```

Parameters

dwTraceID
The handle returned by the calling service or application's initial **TraceRegister** call.

lpszFormat
Pointer to a null-terminated string containing **printf**-style format control information.

arglist
A prepared list of **printf()**-style arguments that define the format of **TraceVprintf** output.

Return Values

If the function succeeds, **TraceVprintf** returns the number of characters output, excluding the terminating null-character.

If the function fails, the return value is zero. This may also indicate that tracing is disabled in the registry. See *Tracing Configuration* for more information.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

See Also

About Tracing, Tracing Reference, **TraceVprintfEx**, **TraceRegister**, **TraceDeregister**, **TracePrintf**, **TracePuts**, **TraceDump**

TraceVprintfEx

Functionality of **TraceVprintfEx** is very similar to that of **TracePrintfEx**, except that it takes a prepared variable argument list as its third variable. **TraceVprintfEx** differentiates itself from **TraceVprintf** in its ability to customize output through the use of non-zero flags. See ***TracePrintfEx*** for more information:

```
DWORD TraceVprintfEx(
  IN DWORD dwTraceID,
  IN DWORD dwFlags,     // OPTIONAL
  IN LPCTSTR lpszFormat,
  IN va_list arglist
);
```

Parameters

dwTraceID
The handle returned by the calling service or application's initial **TraceRegister** call.

dwFlags
Flags that control appearance of **TraceVprintfEx** output. Ensure *dwFlags* is one or more of the following:

TRACE_NO_STDINFO
Suppresses output of the standard information associated with *dwTraceID.*

TRACE_USE_MASK
Determines whether file and/or console output will be generated by comparing the high-order word of *dwFlags* against registry values **FileTracingMask** and **ConsoleTracingMask**.

lpszFormat
Pointer to a null-terminated string containing **printf**-style format control information.

arglist
A prepared list of **printf()**-style arguments that define the format of **TraceVprintf** output.

Return Values

If the function is succesful, **TraceVprintfEx** returns the number of characters output, excluding the terminating null-character.

If the function fails, the return value is zero. This may indicate that tracing is disabled in the registry. See *Tracing Configuration* for more information.

Requirements

Windows NT/2000: Requires Windows 2000. Available as a redistributable for Windows NT 4.0.
Header: Declared in Rtutils.h.
Library: Use Rtutils.lib.

See Also

About Tracing, Tracing Reference, **TraceVprintf**, **TraceRegisterEx**, **TracePrintfEx**, **TracePutsEx**, **TraceDumpEx**

INDEX

Networking Services Programming Elements – Alphabetical Listing

This final part, found in each volume in the Networking Services Library, provides a comprehensive programming element index that has been designed to make your life easier.

Rather than cluttering the TOCs of each individual volume in this library with the names of programming elements, I've relegated such per-element information to a central location: the back of each volume. This index points you to the volume that has the information you need, and organizes the information in a way that lends itself to easy use.

Also, to keep you as informed and up-to-date as possible about Microsoft technologies, I've created (and maintain) a live Web-based document that maps Microsoft technologies to the locations where you can get more information about them. The following link gets you to the live index of technologies:

www.iseminger.com/winprs/technologies

The format of this index is in a constant state of improvement. I've designed it to be as useful as possible, but the real test comes when you put it to use. If you can think of ways to make improvements, send me feedback at *winprs@microsoft.com*. While I can't guarantee a reply, I'll read the input, and if others can benefit, I will incorporate the idea into future libraries.

Locators are arranged by Volume Number followed by Page Number.

A

B

C

N

O

P

Q

R

S

Y

Thank you for acquiring this Microsoft® MSDN™ Universal Subscription. You are eligible to receive a rebate by mail on this product.

To receive your rebate, simply fill out the coupon below and return it along with required proof of purchase to Microsoft. Offer expires December 31, 2000. Coupons must be received by January 31, 2001.

The Microsoft **MSDN Universal Subscription** makes it easy to take advantage of the latest Microsoft tools and technologies. You'll get all the Microsoft operating systems (including client and server platforms), SDKs, DDKs, all the Visual Studio® tools, the BackOffice® Test Platform and Microsoft Office® Developer 2000. Plus, you'll stay ahead of the curve with early releases, service packs, betas, and updates for a full year – automatically! You will also get exclusive, online access to subscription content and updates. MSDN Universal is a timely, convenient, comprehensive resource for developers.

http://msdn.microsoft.com/subscriptions/

MSDN Universal Subscription:

Feature	Benefit
MSDN Library (updated quarterly)	**More than 1.5 GB of programming information and sample code, plus extensive keyword indexing and full-text search engine.**
Complete set of Microsoft operating systems, SDKs, and DDKs	**Includes Microsoft Windows® 98, Microsoft Windows NT® Workstation, Microsoft Windows NT Server, and Microsoft Windows 2000, software development kits (SDK), and driver development kits (DDK).**
Microsoft Visual Studio 6.0 Enterprise Edition	**Includes Visual Studio 6.0, the complete suite of tools to create solutions using Microsoft technologies.**
Microsoft BackOffice Test Platform family	**Develop and test distributed solutions with the BackOffice Test Platform server products and applications.**
Microsoft Office Developer Edition	**Get all the essential tools for building and deploying solutions with Office.**
Updates	**Includes Service Packs, betas, and other product releases for a full year.**

To receive your U.S. $200* mail-in rebate, follow each of the steps below.

*Canadian consumers will receive a check funded in U.S. currency, which will be converted to, and paid in, Canadian funds. The conversion will be calculated by reference to the exchange rate at the time the check is deposited at a financial institution.

1. **Get an MSDN™ Universal Subscription.**
2. **If purchased from a Microsoft reseller, enclose proof of purchase from the MSDN Universal Subscription you acquired.** Eligible proof of purchase is the product box top, with the product name and bar code clearly identified.
3. **Enclose a copy of your dated sales receipt (with date and store name clearly identified)** for the MSDN Universal Subscription you just acquired, OR the packing slip from your initial shipment (if you purchased direct from Microsoft) indicating price paid.
4. **Print your name, address, and phone number here:**

First name   Last name

Company name (if company licenses product)

Mailing address (sorry, no PO boxes)

City   State/Province   ZIP/Postal code   Country

Daytime phone, including area code (in case we have a question about your rebate)

Retailer (store) where MSDN Universal Subscription was acquired   City   State/Province

5. **Mail completed rebate coupon and all required proof of purchase to:**

MSDN Universal Subscription
Promotion #497-00-675
P.O. Box 1140
Ridgely, MD 21681

Microsoft®

In the United States and Canada, if you have questions about this offer, call (800) 622-4445 (8:30 A.M. to 5:30 P.M. eastern time, except weekends and holidays). No rebates will be authorized over the phone.

Please allow 6 to 8 weeks for delivery of your rebate. This offer allows one rebate of U.S. $200* per coupon. Offer good in the 50 United States, the District of Columbia, and Canada only. Offer not valid in U.S. Territories, including Puerto Rico, U.S. Virgin Islands, and Guam. Offer not valid where prohibited, taxed, or restricted by law. OFFER EXPIRES DECEMBER 31, 2000. Coupons must be received by January 31, 2001. Only original coupons will be accepted. Rebate is not valid: if the product was acquired directly from Microsoft and amount of rebate was deducted at time of purchase; in conjunction with other Microsoft offers or rebates; or for upgrades from or on Academic Edition or Not-for-Resale products, or Microsoft products pre-installed or supplied by a manufacturer. Rebate is for Full Package Product MSDN Universal products only. Rebate is good for new subscribers only. Cash redemption value 1/100 of 1¢. Limit one rebate per address.

☐ Occasionally, we offer non-Microsoft products and services to our customers. If you do not wish to receive them, please check here.

Part # 098-86742

Remote Access Services

This essential reference book is part of the five-volume NETWORKING SERVICES DEVELOPER'S REFERENCE LIBRARY. *In its printed form, this material is portable, easy to use, and easy to browse—a highly condensed, completely indexed, intelligently organized complement to the information available on line and through the Microsoft Developer Network (MSDN™). Each book includes an overview of the five-volume library, an appendix of programming elements, an index of referenced Microsoft® technologies, and tips on how and where to find other Microsoft developer reference resources you may need.*

Remote Access Services

This volume focuses on programming guides and reference materials associated with remote access. It covers Remote Access Service (RAS) and the remote access capabilities built into Routing and Remote Access Service (RRAS), which Microsoft Windows NT® Server 4.0 and Microsoft Windows® 2000 Server support. The RAS API and the remote access components built into the RRAS API let you create applications to connect a remote client computer to a Local Area Network (LAN) and enable you to implement a virtual private network (VPN) so that remote computers can participate on the network as if they're connected locally.

Part No. 097-0002786