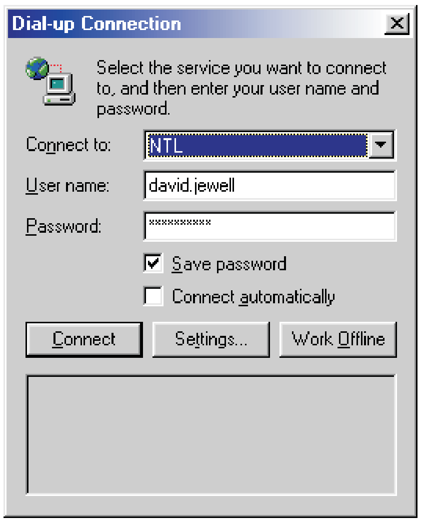
Beating The System: Wrapping The RAS Services API

by Dave Jewell


Dial-up Connection

Select the service you want to connect to, and then enter your user name and password.

Connect to: NTL
User name: david.jewell
Password: **********
[✓] Save password
[ ] Connect automatically

➤ *Figure 1: From where does this familiar dialog get its list of RAS services, usernames and passwords? All is revealed!*

Having spent the last two or three months writing a component that encapsulates some of the Windows file system, this month and next I'm going to concentrate on the development of a couple of non-visual components that 'wrap' the RAS (Remote Access Service) API routines provided by Windows. Amongst other things, this will allow us to programmatically add new entries to the Windows phonebook and monitor the state of current connections. In fact, the extended RAS API routines under Windows 2000 will even allow us to acquire connection statistics.

That said, this month's column is largely a tutorial on how to wrap an API set as a Delphi object, without getting any unpleasant surprises when moving to other platforms, and showing how to fail gracefully if the required API does not exist. In other words, how not to blow up with a GPF if a DLL isn't found. This is especially important when creating software that's got to run on Windows 95, 98, NT, 2000 and so on. Nobody wants to strew their code with dozens of messy 'if this then that' statements according to the Windows platform you're currently running on. Sadly, we can't avoid a certain amount of messiness inside our RAS components, but at least we can do a reasonable job of hiding it from the application program.

But First, What's RAS?

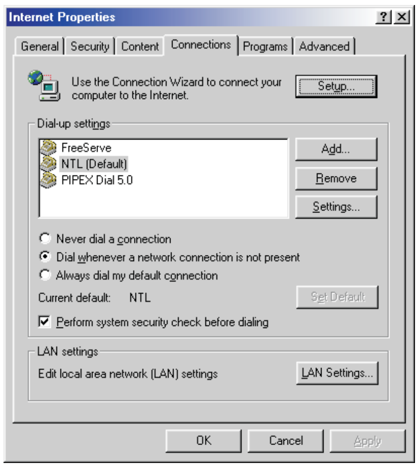
If you look up RAS in the MSDN documentation, it will tell you that the *'Remote Access Service lets users at remote locations work as if connected directly to a computer network, accessing one or more RAS servers'*, which is more than a little vague. A better definition would be to say that the RAS API provides us with a way to programmatically access the DUN (Dial-Up Networking) system, providing direct access to the various defined connections (also called phonebook entries) and enabling us to dial out, hang up, add and delete entries and many other wonderful things.

To put this in concrete terms, click on the `Settings...` button (Figure 1) next time you see this dialog and you'll see the `Connections` tab of the `Internet Properties` dialog (Figure 2). The RAS API allows us to programmatically add, remove and edit the various phone book entries, just as the end user can do from this dialog.

It's rather surprising, given that the RAS API has been around for quite a while, that there aren't any RAS wrapper components in Delphi 5. You won't even find the API-level definitions amongst the contents of \DELPHI5\SOURCE\RTL\WIN. Perhaps Borland thought the API was too convoluted to be worth bothering with. I thought that too, from time to time, but decided to plough on regardless!

The RAS components I've developed are all derived from a common ancestor called `TRASBaseComponent`, the source code for which is shown in Listing 1. I did things like this because there's a small amount of code that's needed whatever RAS services you're using, and it makes sense to centralise this in one place. As you can see from the code, `TRASBaseComponent` attempts to load two DLLs, the first of which is RASAPI32.DLL. Under Windows 95, 98, NT 4.0 and Windows 2000, this DLL contains most of the RAS API routines that are available to applications... but not necessarily all! At some point, Microsoft decided to add some more RAS API calls and, rather than adding them to a new release of RASAPI32.DLL, they stuffed them into another DLL called RNAPH.DLL. Later, they *did* add the new routines to a subsequent release of RASAPI32.DLL, thus rendering RNAPH.DLL obsolete. So (if Microsoft's documentation can be believed) some early versions of Windows might not have this

➤ *Figure 2: With a good grasp of the RAS API and a set of Delphi components to wrap up all Microsoft's run-away complexity, you can design your own custom editors for phonebook entries.*


Internet Properties

Use the Connection Wizard to connect your computer to the Internet.

Dial-up settings
FreeServe
NTL (Default)
PIPEX Dial 5.0

( ) Never dial a connection
(●) Dial whenever a network connection is not present
( ) Always dial my default connection

Current default: NTL

[✓] Perform system security check before dialing

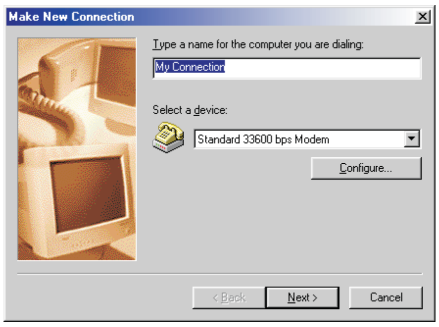
LAN settings
Edit local area network (LAN) settings

➤ *Listing 1: Facing page.*

```
unit RASControls;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs;
type
  TRasDialParams = record
    dwSize: Integer;
    EntryName: array [0..256] of Char;
    PhoneNumber: array [0..128] of Char;
    CallbackNumber: array [0..128] of Char;
    UserName: array [0..256] of Char;
    Password: array [0..256] of Char;
    Domain: array [0..15] of Char;
  end;
  TRasDialParamsNT4_2000 = record
    dwSize: Integer;
    EntryName: array [0..256] of Char;
    PhoneNumber: array [0..128] of Char;
    CallbackNumber: array [0..128] of Char;
    UserName: array [0..256] of Char;
    Password: array [0..256] of Char;
    Domain: array [0..15] of Char;
    SubEntry: Integer;
    CallbackId: Integer;
  end;
  TRasEntry = record
    dwSize, Options: Integer;        // General stuff
    CountryID, CountryCode: Integer; // Phone/Country info
    AreaCode: array [0..10] of Char;
    LocalPhoneNumber: array [0..128] of Char;
    AlternateOffset: Integer;
    ipaddr, ipaddrDns, ipaddrDnsAlt,
      ipaddrWins, ipaddrWinsAlt: Integer;        // PPP/Ip
    FrameSize, NetProtocols,
      FramingProtocol: Integer;                  // Framing
    Script: array [0..Max_Path-1] of Char;       // Scripting
    AutodialDll: array [0..Max_Path-1] of Char;  // AutoDial
    AutodialFunc: array [0..Max_Path - 1] of Char;
    DeviceType: array [0..16] of Char;           // Device
    DeviceName: array [0..128] of Char;
    X25PadType: array [0..32] of Char;           // X.25
    X25Address: array [0..200] of Char;
    X25Facilities: array [0..200] of Char;
    X25UserData: array [0..200] of Char;
    Channels: Integer;
    Reserved1, Reserved2: Integer;
  end;
  TRASBaseComponent = class (TComponent)
  private
    fErrorText: String;
    fError: Integer;
    fOnError: TNotifyEvent;
    fRasLib, fRasExtensionsLib: THandle;
    fWin2000, fWinNT4, fAvailable, fDummy1: Boolean;
    function GetProc (ProcName: PChar): Pointer;
    function CallProc (Err: Integer): Boolean;
  public
    constructor Create (AOwner: TComponent); override;
    destructor Destroy; override;
    property ErrorText: String read fErrorText;
    property Error: Integer read fError;
  published
    property Available: Boolean read fAvailable
      write fDummy1 stored False;
    property OnError: TNotifyEvent read fOnError
      write fOnError;
  end;
  TRASPhoneBookManager = class(TRASBaseComponent)
  private
    fItemIndex: Integer;
    fEntries, fDummy2: TStrings;
    fPhoneBookFileName, fDummy3: String;
    procedure SetItemIndex (Value: Integer);
    function PhoneBookNameAsPChar: PChar;
    procedure SetPhoneBookFileName (const Value: String);
    function GetDialParameters (Index: Integer): String;
    function GetEntryProperties (Index: Integer): String;
    function InternalGetDialParameters(
      var Dest: TRasDialParamsNT4_2000;
      var GotPassword: Bool): Integer;
    function InternalGetEntryProperties(
      var Dest: TRASEntry): Boolean;
  protected
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    function Add: Boolean;
    function Delete: Boolean;
    function Edit: Boolean;
    function Rename(const NewName: String): Boolean;
    function ValidateEntryName(const EntryName: String):
      Boolean;
    procedure Refresh;
  published
    property ItemIndex: Integer read fItemIndex
      write SetItemIndex stored False;
    property PhoneBookFileName: String
      read fPhoneBookFileName write SetPhoneBookFileName;
    property Entries: TStrings read fEntries
      write fDummy2 stored False;
    property UserName: String index 0 read GetDialParameters
      write fDummy3 stored False;
```

```
    property Password: String index 1
      read GetDialParameters write fDummy3 stored False;
    property PhoneNumber: String index 0
      read GetEntryProperties write fDummy3 stored False;
    property DeviceType: String index 1
      read GetEntryProperties write fDummy3 stored False;
    property DeviceName: String index 2
      read GetEntryProperties write fDummy3 stored False;
  end;
procedure Register;
implementation
constructor TRASBaseComponent.Create (AOwner: TComponent);
begin
  Inherited Create (AOwner);
  fRasLib := LoadLibrary ('rasapi32.dll');
  fRasExtensionsLib := LoadLibrary ('rnaph.dll');
  fAvailable := fRasLib <> 0;
  fWin2000 := (Win32Platform = Ver_Platform_Win32_NT) and
    (Win32MajorVersion >= 5);
  fWinNT4 := (Win32Platform = Ver_Platform_Win32_NT) and
    (Win32MajorVersion = 4);
end;
destructor TRASBaseComponent.Destroy;
begin
  if fRasLib <> 0 then FreeLibrary(fRasLib);
  if fRasExtensionsLib <> 0 then
    FreeLibrary(fRasExtensionsLib);
  Inherited;
end;
function TRASBaseComponent.GetProc(ProcName: PChar):
  Pointer;
begin
  Result := Nil;
  if fAvailable then begin
    Result := GetProcAddress (fRasLib, ProcName);
    if (Result = Nil) and (fRasExtensionsLib <> 0) then
      Result := GetProcAddress(fRasExtensionsLib,ProcName);
  end;
end;
function TRASBaseComponent.CallProc(Err: Integer): Boolean;
var
  szErr: array [0..1024] of Char;
  RasGetErrorString: function(Err: Integer; Buff: PChar;
                     BuffSize: Integer): Integer; stdcall;
begin
  fError := Err;
  Result := Err = 0;
  if Result then fErrorText := ''
  else begin
    RasGetErrorString := GetProc ('RasGetErrorStringA');
    if RasGetErrorString(Err, szErr, sizeof(szErr))=0 then
      fErrorText := szErr
    else begin
      fErrorText := SysErrorMessage (Err);
      if fErrorText = '' then
        fErrorText := Format ('Unknown error (%d)', [Err]);
    end;
    fErrorText := 'RAS: ' + fErrorText;
    if Assigned(fOnError) then fOnError(Self);
  end;
end;
constructor TRASPhoneBookManager.Create(
  AOwner: TComponent);
begin
  Inherited Create (AOwner);
  fEntries := TStringList.Create;
  Refresh;
end;
destructor TRASPhoneBookManager.Destroy;
begin
  fEntries.Free;
  Inherited;
end;
procedure TRASPhoneBookManager.SetPhoneBookFileName(
  const Value: String);
begin
  if (fPhoneBookFileName <> Value)
    and FileExists (Value) then begin
    fPhoneBookFileName := Value;
    Refresh;
  end;
end;
function TRASPhoneBookManager.PhoneBookNameAsPChar: PChar;
begin
  if fPhoneBookFileName = '' then Result := Nil
  else Result := PChar(fPhoneBookFileName);
end;
procedure TRASPhoneBookManager.Refresh;
type
  TRasEntryName = record
    dwSize: Integer;
    szEntryName: array [0..257] of Char;
  end;
  TRasEntryName2000 = record
    dwSize: Integer;
    szEntryName: array [0..257] of Char;
    dwFlags: Integer;
    szPhonebookPath: array [0..Max_Path] of Char;
  end;
```

```
var
  CurEntry, Buffer: PChar;
  Idx, BufSize, NumEntries, EntrySize: Integer;
  RasEnumEntries: function(Reserved,Phonebook,Buffer: PChar;
   var BufSize,NumEntries: Integer): Integer; stdcall;
begin
  if fAvailable then begin
    fEntries.Clear;
    RasEnumEntries := GetProc('RasEnumEntriesA');
    EntrySize := sizeof (TRasEntryName);
    if fWin2000 then EntrySize := sizeof(TRasEntryName2000);
    Idx := EntrySize;
    BufSize := sizeof (Idx);
    RasEnumEntries(Nil, PhoneBookNameAsPChar, @Idx,
      BufSize, NumEntries);
    Buffer := AllocMem (BufSize);
    try
     PInteger (Buffer)^ := EntrySize;
     if CallProc (RasEnumEntries (Nil, PhoneBookNameAsPChar,
       Buffer, BufSize, NumEntries)) then begin
       CurEntry := Buffer;
       for Idx := 0 to NumEntries - 1 do begin
         fEntries.Add (CurEntry + sizeof (Integer));
         Inc (CurEntry, EntrySize);
       end;
     end;
     finally
       FreeMem (Buffer);
       if fEntries.Count > 0 then fItemIndex := 0
       else  fItemIndex := -1;
     end;
  end;
end;
procedure TRASPhoneBookManager.SetItemIndex(
  Value: Integer);
begin
  if (Value >= 0) and (Value < fEntries.Count) and
    (fEntries.Count > 0) then
    fItemIndex := Value;
end;
function TRASPhoneBookManager.Add: Boolean;
var
  RasCreatePhoneBookEntry: function(WndParent: hWnd;
    Phonebook: PChar): Integer; stdcall;
begin
  Result := False;
  if fAvailable then begin
    RasCreatePhoneBookEntry :=
      GetProc('RasCreatePhonebookEntryA');
    if Assigned (RasCreatePhoneBookEntry) then
      Result := CallProc(RasCreatePhoneBookEntry(
        Application.Handle, PhoneBookNameAsPChar));
    if Result then Refresh;
  end;
end;
function TRASPhoneBookManager.Delete: Boolean;
var
  RasDeleteEntry: function(Phonebook, EntryName: PChar):
    Integer; stdcall;
begin
  Result := False;
  if fAvailable and (fItemIndex >= 0) then begin
    RasDeleteEntry := GetProc ('RasDeleteEntryA');
    if Assigned (RasDeleteEntry) then
       Result := CallProc(RasDeleteEntry(
         PhoneBookNameAsPChar,
         PChar(fEntries[fItemIndex])));
    if Result then Refresh;
  end;
end;
function TRASPhoneBookManager.Edit: Boolean;
var
  RasEditPhonebookEntry: function(WndParent: hWnd;
    Phonebook, EntryName: PChar): Integer; stdcall;
begin
  Result := False;
  if fAvailable and (fItemIndex >= 0) then begin
    RasEditPhonebookEntry :=
      GetProc('RasEditPhonebookEntryA');
    if Assigned (RasEditPhonebookEntry) then
      Result := CallProc(RasEditPhonebookEntry(
        Application.Handle, PhoneBookNameAsPChar,
        PChar(fEntries [fItemIndex])));
    if Result then Refresh;
  end;
end;
function TRASPhoneBookManager.ValidateEntryName(
  const EntryName: String): Boolean;
var
  RasValidateEntryName: function(Phonebook, EntryName:
                        PChar): Integer; stdcall;
begin
  Result := False;
  if fAvailable then begin
    RasValidateEntryName :=
      GetProc('RasValidateEntryNameA');
  if Assigned (RasValidateEntryName) then
    Result := CallProc (RasValidateEntryName(
      PhoneBookNameAsPChar, PChar (EntryName)));
  end;
end;
function TRASPhoneBookManager.Rename(
  const NewName: String): Boolean;
var
  RasRenameEntry: function(Phonebook, OldName, NewName:
                  PChar): Integer; stdcall;
begin
  Result := False;
  if fAvailable and (fItemIndex >= 0) and
     ValidateEntryName(NewName) then begin
    RasRenameEntry := GetProc('RasRenameEntryA');
    if Assigned (RasRenameEntry) then
      Result := CallProc(RasRenameEntry(
        PhoneBookNameAsPChar, PChar(fEntries[fItemIndex]),
        PChar(NewName)));
    if Result then Refresh;
  end;
end;
function TRASPhoneBookManager.InternalGetDialParameters(
  var Dest: TRasDialParamsNT4_2000; var GotPassword: Bool):
  Integer;
var
  RasGetEntryDialParams: function(Phonebook: PChar;
    var RasDialParams: TRasDialParamsNT4_2000;
    var GotPassword: Bool): Integer; stdcall;
begin
  Result := 0;
  if fAvailable and (fItemIndex >= 0) then begin
    RasGetEntryDialParams :=
      GetProc('RasGetEntryDialParamsA');
    if Assigned (RasGetEntryDialParams) then begin
      if fWin2000 or fWinNT4 then
        Dest.dwSize := sizeof(TRasDialParamsNT4_2000)
      else
        Dest.dwSize := sizeof (TRasDialParams);
      StrPCopy(Dest.EntryName, fEntries[fItemIndex]);
      if CallProc(RasGetEntryDialParams(
        PhoneBookNameAsPChar, Dest, GotPassword)) then
        Result := Dest.dwSize;
    end;
  end;
end;
function TRASPhoneBookManager.GetDialParameters(
  Index: Integer): String;
var
  GotPassword: Bool;
  Params: TRasDialParamsNT4_2000;
begin
  if InternalGetDialParameters(Params,GotPassword) > 0
    then begin
    if not GotPassword then
      Params.Password := '---not available----';
    case Index of
      0 : Result := Params.UserName;
      1 : Result := Params.Password;
    end;
  end;
end;
function TRASPhoneBookManager.InternalGetEntryProperties(
  var Dest: TRASEntry): Boolean;
var
  EntrySize, DevInfoSize: Integer;
  Buffer: array [0..10000] of Char;
  RasGetEntryProperties: function(Phonebook, EntryName:
    PChar; var Entry; var EntrySize: Integer; DevInfo:
    Pointer; var DevInfoSize: Integer): Integer; stdcall;
begin
  Result := False;
  if fAvailable and (fItemIndex >= 0) then begin
    RasGetEntryProperties :=
      GetProc('RasGetEntryPropertiesA');
    if Assigned (RasGetEntryProperties) then begin
      PInteger(@Buffer)^ := sizeof(TRASEntry);
      EntrySize := sizeof(Buffer);
      DevInfoSize := 0;
      Result := CallProc(RasGetEntryProperties(
        PhoneBookNameAsPChar, PChar(fEntries[fItemIndex]),
        Buffer, EntrySize, Nil, DevInfoSize));
      if Result then
        Move (Buffer, Dest, sizeof(TRASEntry));
    end;
  end;
end;
function TRASPhoneBookManager.GetEntryProperties(
  Index: Integer): String;
var Props: TRASEntry;
begin
  if InternalGetEntryProperties (Props) then begin
    case Index of
      0 : Result := Props.LocalPhoneNumber;
      1 : Result := Props.DeviceType;
      2 : Result := Props.DeviceName;
    end;
  end;
end;
procedure Register;
begin
  RegisterComponents('DelphiMag', [TRASPhoneBookManager]);
end;
end.
```



➤ *Figure 3: The Make New Connection wizard is what appears when you invoke the RasCreatePhoneBookEntry API call. The various RAS-related dialogs are implemented in the undocumented RNAUI.DLL.*

supplementary DLL at all, whereas more recent versions might. I've discovered RNAPH.DLL on my Windows 98 (First Release) system, but it doesn't exist on Windows 2000.

So what's the best way out of this nightmare? Given a particular API call, how do we track it down? I decided the simplest approach was to check RASAPI32.DLL first and only check RNAPH.DLL if the routine doesn't exist in the first library. That way, things should work properly on systems that don't require the extra DLL, and on systems that do.

You can see how this works out in the `TRASBaseComponent.GetProc` routine. Given an API routine name, it first checks the main API library, and only checks the secondary library if necessary. The destructor for `TRASBaseComponent` automatically ensures that the libraries are unloaded when the component is destroyed, and the constructor also takes care of setting up a `Boolean` property, `Available`, which indicates whether or not RAS services are available. This read-only property uses a dummy `write` member to fool the property inspector into displaying it. Notice the property is defined with the `stored false` attribute, because it doesn't make sense for it to be persistent. There are also two internal flags, `fWin2000` and `fWinNT4`, which are needed in various places to cope with certain platform-specifics.

The final job of `TRASBaseComponent` is to provide a centralised place for error handling. Whenever one of the RAS API routines is called, the function result is passed to the `CallProc` method. If the error code was zero, then this sets the public property `ErrorText` to an empty string. If a non-zero error code was returned, the code uses the `RasGetErrorString` to translate the RAS error code into something that's human readable, and places the result in `ErrorText`. In addition, we maintain another property, `Error`, for clients who'd prefer to see the original error code. Finally, there's an `OnError` event handler which is invoked whenever something goes wrong.

Fun With Phonebooks.

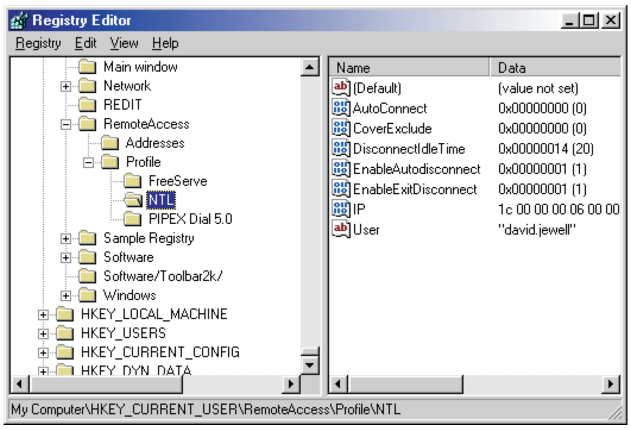
I've nothing else to say about `TRASBaseComponent`, so let's turn to something a bit more interesting. As my first foray into RAS API twiddling, I decided to write a component that encapsulates access to the phonebook, called `TRASPhoneBookManager`. As I indicated earlier, the list of available connections are stored in what Microsoft call the phonebook. Under Windows 95 and 98, this is actually a part of the system registry, but under NT (including Windows 2000) the phonebook is a separate disk file with the extension .PBK. Just to make things even more fun, NT supports multiple phonebooks.

Fortunately, it's easy to cater for all this. If you look at the code for `TRASPhoneBookManager`, you'll see there's a `String` property called `PhoneBookFileName`. This will be set to an empty string by default, which is just what we want for W9x, since it will be interpreted as a reference to the registry-based default phone book. If you set this string to point to a valid .PBK file under NT, that phonebook will be used instead. If you specify an empty string under NT, then the default phonebook will be used.

This raises an interesting little subtlety. If you look at the various phonebook-related RAS API functions, you'll see they all take a `PChar` parameter which identifies the phonebook to use. The API documentation states that you should pass `NIL` to invoke the default behaviour. However, what happens if you pass `PChar(Str)` to an API routine where `Str` is an empty string? Does `NIL` get passed? The answer is no, it doesn't! What gets passed is a pointer to a zero byte. I experimented with this under Windows 98 and discovered that the various phonebook routines were happy to accept a pointer to an empty string. In fact, when I peeked at the code inside RASAPI32.DLL, I found that none of the Windows 98 routines even bothered to look at this parameter, it was totally ignored. Even so, in the interests of abiding by the letter of the law, I added a small routine called `PhoneBookNameAsPChar` to ensure `NIL` (`NULL` if you're a C++ person) gets passed when the phonebook filename is empty.

The most important property in this component is `Entries`, which is implemented as a read-only `TStrings` component containing the number of entries in the current phonebook (or in the registry if you're not running NT). This stringlist is set up by the code inside the `Refresh` method which is

➤ *Figure 4: Under Windows 95 and 98, there's only one RAS phonebook, occupying a specific place in the Windows registry. As you can see here, I have three different ISP services currently installed.*



```
        Main window
      + Network
        REDIT
      - RemoteAccess
          Addresses
        - Profile
            FreeServe
            NTL
            PIPEX Dial 5.0
      + Sample Registry
      + Software
        Software/Toolbar2k/
      + Windows
  + HKEY_LOCAL_MACHINE
  + HKEY_USERS
  + HKEY_CURRENT_CONFIG
  + HKEY_DYN_DATA
```

Name	Data
(Default)	(value not set)
AutoConnect	0x00000000 (0)
CoverExclude	0x00000000 (0)
DisconnectIdleTime	0x00000014 (20)
EnableAutodisconnect	0x00000001 (1)
EnableExitDisconnect	0x00000001 (1)
IP	1c 00 00 00 06 00 00
User	"david.jewell"

My Computer\HKEY_CURRENT_USER\RemoteAccess\Profile\NTL called from the component's constructor. I've also made the `Refresh` method public so the application can refresh the phonebook entries from time to time, to ensure it's got the current state of play.

If you're running NT 3.51, or an early version of NT 4.0, then don't be too zealous in calling `Refresh`! This is because the `Refresh` function has to call `RASEnumEntries`, a RAS API routine which was notorious for leaking memory in early implementations. Microsoft have fixed the problem now, but if you expect your code to run on an antique version of NT, then call `Refresh` judiciously!

At the same time, there's an `ItemIndex` property which is used to refer to a specific item within the phonebook. Be careful to set it to the appropriate item, especially before calling the `Delete` method!

The `Refresh` routine is necessarily rather involved, mainly to cope with the brain-dead API routine that it's calling. Despite the name, the `RASEnumEntries` routine doesn't give you phonebook entries one at a time, it gives you the whole lot in one go. Ordinarily, it wouldn't be too much trouble stepping through the memory buffer and accessing the information, but Microsoft have thoughtfully increased the size of individual entries in Windows 2000!

You can see what I mean by consulting the listing. Prior to Windows 2000, a phonebook entry name record consisted of only two fields (see `TRasEntryName`), a record size specifier followed by a character array containing the actual entry name. With the advent of Windows 2000, a couple of new fields got tacked onto the record in order to specify flags (system-wide or per-user) for each phonebook entry and to return the full pathname of the .PBK file for that specific entry. (see `TRasEntryName2000`).

This means that, prior to calling `RASEnumEntries`, you've got to figure out what size data structure you're dealing with; this is why we set up the `fWin2000` field in the ancestor component. Another, er, interesting aspect of the `RASEnumEntries` routine is that you don't know how big to make the buffer to receive the array of records. The best solution is to deliberately give the routine a buffer which is far too small, whereupon it will grudgingly tell you how big it wants the buffer to be. Yes folks, as API routines go, `RASEnumEntries` is about as friendly as a cornered rat.

Once you've got these little obstacles sorted out, the rest is simple. The `GetProc` method is called to obtain the procedure pointer for the API call. In this case I haven't bothered checking to see if we get `NIL` back because this routine was implemented way back in NT 3.1! The `fEntries` stringlist is cleared and `EntrySize` is calculated according to the record size we're working with. After making a dummy call to get the required buffer size, the requisite amount of memory is allocated and the real call is made. The code then simply steps through the array, incrementing by `EntrySize` each time round the loop in order to access successive records.

The `RasCreatePhonebookEntry` routine is used to create a new phonebook entry. Well, sort of. All it *actually* does is call the built-in '*Make New Connection*' wizard, which you can see running in Figure 3. It's rather a shame that there is no API to directly add the information into the registry or current phonebook. At least, not officially, but maybe this is something we'll look at in a future instalment (see *RAS API Secrets*). I wrapped the call `RasCreatePhonebookEntry` with the much easier to use `Add` method: look Ma, no parameters! Like the various other methods discussed here, it returns `True` on success or `False` on failure. If it fails, an application can consult `Error` and `ErrorText` to get the low-down on what went wrong. The only thing worthy of note in the `Add` routine is the use of `Application.Handle` as a sneaky way of providing a parent window handle.

```
function Remote_CreateEntry (hWndParent: hWnd): Integer;
stdcall; external 'rnaui.dll';
function Remote_EditEntry (hWndParent: hWnd; EntryName: PChar): Integer;
stdcall; external 'rnaui.dll';
```

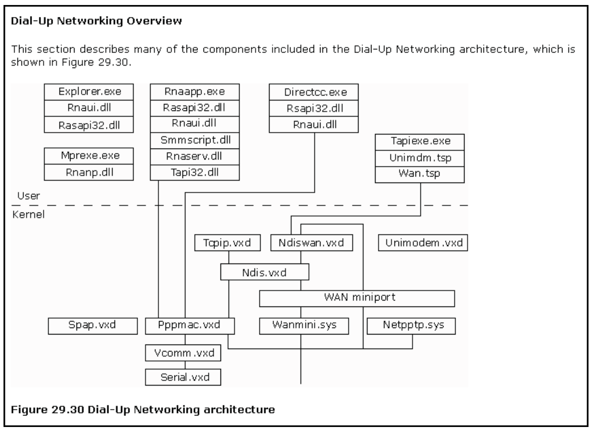
➤ *Above: Listing 2*

➤ *Below: Listing 3*

```
typedef struct _RASDIALPARAMS {
  DWORD  dwSize;
  TCHAR  szEntryName[RAS_MaxEntryName + 1];
  TCHAR  szPhoneNumber[RAS_MaxPhoneNumber + 1];
  TCHAR  szCallbackNumber[RAS_MaxCallbackNumber + 1];
  TCHAR  szUserName[UNLEN + 1];
  TCHAR  szPassword[PWLEN + 1];
  TCHAR  szDomain[DNLEN + 1] ;
#if (WINVER >= 0x401)
  DWORD  dwSubEntry;
  DWORD  dwCallbackId;
#endif
} RASDIALPARAMS;
```

```
property UserName: String index 0
  read GetDialParameters
  write fDummy3 stored False;
property Password: String index 1
  read GetDialParameters
  write fDummy3 stored False;
```

➤ *Listing 4*


Dial-Up Networking Overview

This section describes many of the components included in the Dial-Up Networking architecture, which is shown in Figure 29.30.

Figure 29.30 Dial-Up Networking architecture

➤ *Figure 5: Why use one DLL when half a dozen will do? Microsoft show once again that the KISS principle of software engineering is an alien concept in Redmond.*

Like most of Microsoft's routines which display a dialog, you need to provide a parent window. I got the Application.Handle trick from DIALOGS.PAS.

My implementation of the Delete and Edit methods follows exactly the same pattern. They each work with the current value of ItemIndex, which forms an index into the Entries property, thus identifying the phonebook entry to be deleted or edited. The code checks that ItemIndex is within range, and then invokes GetProc to retrieve the appropriate API pointer from the RAS DLL. In both cases CallProc is used to set the error status information and call OnError if the event handler has been assigned. As with RasCreatePhonebookEntry, the RasEditPhonebookEntry routine brings up the appropriate dialog box from RNAUI.DLL.

For the terminally curious, and those who like living dangerously, the undocumented dialog routines in RNAUI.DLL can easily be called directly without recourse to RASAPI32.DLL (see Listing 2). Well, I'm sure you get the idea!

The RAS API also implements a routine called RasRenameEntry which allows an existing phonebook entry to be renamed. As shown in the listing, I wrapped this up into an easily callable method: Rename. While doing so, I noticed the documentation mentions the presence of another routine called RasValidateEntryName, so I added support for this too, ensuring the new connection name passed to Rename goes through my ValidateEntryName code first. In practice, I don't think that ValidateEntryName does very much, but one may as well play ball.

Delving Deeper

OK, so we can add, delete and rename phonebook entries, but what about the actual business of phone numbers, usernames and (shhh...) passwords? In order to retrieve password and username information for a specific connection, we have to use the snappily-named RasGetEntryDialParams routine. Microsoft are now officially deprecating the use of this routine, telling us that we need to move over to the more recent RasGetCredentials. Well, thanks very much Microsoft, but have you noticed that you didn't bother to implement this routine under Windows 95/98? I'll take my chances and stick with RasGetEntryDialParams for now. By the time Microsoft carry out their threat of removing RasGetEntryDialParams from the RAS API, the Windows 9x product line will no doubt be history... maybe.

RasGetEntryDialParams takes a pointer to a data structure which (in C/C++ terms) looks like Listing 3. As you can see, we're faced with the same problem we encountered with RasEnumEntries: the data structure used is operating system version dependent. For NT 4.0 and Windows 2000, the last two fields exist, whereas for other platforms they don't. The dwSize field has to be set up prior to calling RasGetEntryDialParams, and you also have to set up the szEntryName field with the name of the required entry. The call will then fill in the other fields of the data structure. Somewhat counter-intuitively, we can't use this routine to retrieve the phone number, even though there is a szPhoneNumber field in the structure. For that, we need another routine called RasGetEntryProperties. More on that in a moment.

As you can see from the code, TRASPhoneBookManager implements a couple of string properties, like those shown in Listing 4.

As you change the ItemIndex property, you'll see the UserName and Password properties change in the object inspector: deeply cool. Yes, I'm a great one for making properties 'inspectable' if at all possible (ie published properties which pretend to be read/write so that the object inspector will deign to display them) and, to this end, I've used another dummy variable, fDummy3 as the write clause. Can

RAS API Secrets: To Boldly Go...

Microsoft's RAS API is a delightfully baroque collection of DLLs, registry entries and (in the case of NT) file-based phonebooks. Personally, I'm the sort of guy who would rather cut through the multiple layers of Microsoft code and get right down to 'hitting the metal'. Not only will this make your application a lot faster and less dependent on Microsoft DLLs, but arguably a lot more reliable too ☺. That said, it should be obvious that riding rough-shod over the official APIs might buy you ultimate flexibility, but perhaps ultimate disaster if the underlying implementation ever changes.

If you do want to venture into the unknown, consider the simple process of programmatically creating a new phonebook entry. As indicated elsewhere, the RasCreatePhonebookEntry API call simply invokes the new connection wizard. The RasCreatePhonebookEntry code does this by loading the undocumented RNAUI.DLL which contains much of the user interface code involved. This in turn is responsible for making persistent registry changes (on Windows 95/98) according to whatever's typed into the various dialogs. If you consult Figure 4, you'll see the registry sub-tree at HKEY_CURRENT_USER\RemoteAccess. If you want to create a new, empty phonebook entry, just try adding a string value (eg 'Wombat') to the Addresses key and, hey presto, you'll find that if you then open the Dial-Up Networking folder under My Computer, Wombat will magically appear. Of course, there's a lot more than this to do in order to get a valid phonebook entry, I'm simply making the point that, ultimately, it all comes down to registry manipulation under Windows 95/98.

Under NT 4.0 and Windows 2000, phonebook entries reside in disk files. Interestingly, there are a whole slew of undocumented routines inside the Windows 2000 version of RASAPI32.DLL which relate to manipulation of these .PBK files, 23 of them no less! With names like RasFileLoad, RasfileGetSectionName, RasFileWrite, etc, it's obvious that these are the low-level manipulation routines for NT phonebooks. Hmmm... GetSectionName sounds an awful lot like an .INI file, doesn't it? Surely not? Oh yes! No prizes for guessing what the inside of a .PBK file looks like ☺.

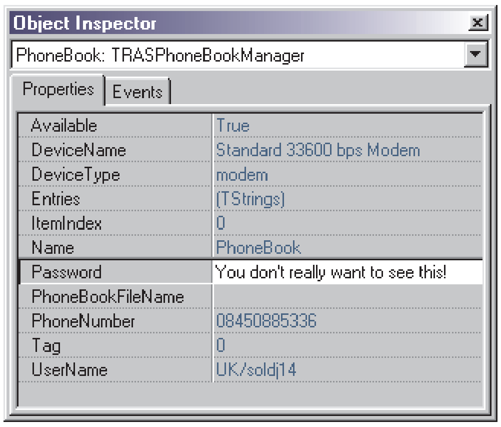
anyone explain to me why this is OK, even though both `UserName` and `Password` are indexed properties? I expected to have to provide a dummy 'writer' routine, which takes an index value and string as parameters, but the Delphi 3 compiler accepts this construction without comment. Either it's smart enough to realise that I only want to use indexing to read the property, or too dim to spot my misdemeanour, I'm not sure which! Anyway, it works. Then again, I've only tested this code with Delphi 3 so far: your mileage may vary.

The `GetDialParameters` code is relatively straightforward because most of the work is done in a helper routine called `InternalGetDialParameters`. It takes an index value and returns the appropriate string after retrieving a valid parameter block. One wrinkle is that, depending on security considerations, etc, you may or may not be able to retrieve the password of a specific account, and if the code detects that no password was retrieved, then the `Password` property is set to —`not available`— by default.

Yes, Dave, but what about getting the phone number? As I mentioned, we need another routine called `RasGetEntryProperties` to get the phone number of a particular phonebook entry. The bad news is that this routine requires an absolutely horrendous data structure, `TRasEntry`, which you'll find near the beginning of the listing.

I hope that this isn't the case, but one or two readers might feel disposed to berate me for getting rid of all Microsoft's symbolic constants in the various structures presented in this article. Why have I replaced `RAS_MaxAreaCode` with 10, `RAS_MaxPhoneNumber` with 128, and so on. Well, for starters, I think that the RAS API is quite complex enough already without cluttering things up with dozens of extra constants. I don't like obfuscating my code and I don't anticipate that Microsoft will change any of these constants any time soon. In any event the proper place for all these constants and structure definition is in a RAS.PAS (sounds good, that!) file supplied by Borland. And, yes guys, that was a big hint. Maybe in Delphi 6?

➤ *Figure 6: Here's the phonebook manager component as seen in the Object Inspector.*



Object Inspector

PhoneBook: TRASPhoneBookManager

Properties | Events

Available	True
DeviceName	Standard 33600 bps Modem
DeviceType	modem
Entries	(TStrings)
ItemIndex	0
Name	PhoneBook
Password	You don't really want to see this!
PhoneBookFileName	
PhoneNumber	08450885336
Tag	0
UserName	UK/soldj14

As a final little smack in the face from Microsoft (what, me, bitter and twisted?), the `RasGetEntryProperties` call can potentially return a number of alternative phone numbers appended to the end of the `TRasEntry` data structure. Thus, you've got to be sure to allocate a buffer big enough to cope. In this relatively simple phonebook implementation, I've ignored such niceties as alternative phone numbers: after all, what I'm really trying to do here is demonstrate how to wrap an API set in a platform independent manner. It's up to you to flesh out the bones with whatever 'meat' you're most interested in, so to speak! Nevertheless, you do need to allocate some space after the `TRasEntry` structure for those extra phone numbers.

You can see how I've tackled this in the `InternalGetEntryProperties` routine. The buffer allocated is actually 10,000 bytes long (what's 32-bit programming for if you can't casually throw 10Kb on the stack?) and any extra phone numbers will go into this buffer after the 'real' `TRASEntry` data structure. Once the data has been successfully read, then we just pass back `sizeof(TRasEntry)` bytes to the caller.

Armed with the `InternalGetEntryProperties` routine, I added three more indexed properties to the phonebook component to return phone number, device type (modem, X.25, etc) and device name associated with this entry. If you're interested in those other X.25-related strings in `TRasEntry`, it would be trivially simple to expose them as properties.

Maybe you're thinking that this component isn't enormously efficient, because it hits the RAS API every time Delphi reads one of those indexed properties. Well, that's true, but I wanted the component to keep in sync with the current state of the phonebook where possible and, besides, the latency of these routines is pretty low. On Windows 9x we're merely accessing the registry, and on NT we're accessing an .INI file, er, I mean a .PBK file.

That's all we've got time for this month, but in the next instalment I'll continue 'Rapping RAS' with some more enhancements to the phonebook component, which will include the ability to modify some of these read-only properties such as password, username and phone number.

At the same time, I'll be presenting the code for a connection manager which automatically notifies a Delphi application when a RAS connection has been created or destroyed and, who knows, we might even have time to explore the innards of those .PBK files.

Warning: because of tight deadlines (and a literally thunderstruck modem!) so far I've only tested this code under Windows 98. If you want to start using RAS under Windows 2000 and can't wait for next month's code (and possible amendments) do proceed with caution.

Dave Jewell is a freelance consultant/programmer and technical journalist specialising in system-level Windows and DOS work. He is Technical Editor of *Developers Review* which is also published by iTec. You can contact Dave as TechEditor@itecuk.com