The Pascal unit you'll use to create our Delphi RAPI application is a freeware unit, generously provided by Scott Crossen. Crossen created a Pascal-/Delphi-based import of all of the functions, types, and constants listed in `Rapi.h`. This in itself is no small feat—many of the structures in `Rapi.h` are quite complex and were probably very difficult to port. But the best thing about the `Rapi.pas` unit that Crossen created is that all of the functions are loaded dynamically.

In C/C++, when you use the default `Rapi.h`, the compiler statically links `Rapi.lib` into your program. When the program starts on the user's machine and they're not running CE Services, it will crash before it even gets off the ground.

What Crossen did differently was to attempt to find and load `Rapi.dll` at runtime. If you can't find `Rapi.dll`, you exit gracefully and return an error code. In contrast to linking in `rapi.lib`, Crossen's implementation first checks to see if RAPI is available. If RAPI isn't available on the user's machine, the program won't crash; it can still perform every non-RAPI feature just fine.

If you look at the code to do this, you'll see it starts with a call to `LoadLibrary()`, passing in `'RAPI.DLL'` as the library to load:

```
RapiModule := LoadLibrary('RAPI.DLL');
```

Check to make sure that the library was successfully loaded:

```
if RapiModule > HINSTANCE_ERROR then
begin
  Result := True;
```

Next, proceed to retrieve the addresses of all of the RAPI functions, one at a time:

```
    @mCeRapiInit := GetProcAddress(RapiModule, 'CeRapiInit');
    //...and repeat for each RAPI function
    @mCeGetSystemPowerStatusEx:=  GetProcAddress(RapiModule, 'CeGet-
SystemPowerStatusEx');

  end
  else
    Result := False;
end;
```

You've managed to successfully dynamically load all of the RAPI functions at once. The amazing thing about the "any Desktop development tool" aspect of

RAPI is that the code to retrieve all of the databases looks virtually the same in Pascal as it does in C/C++.

For instance, when you retrieve the list of databases in your Delphi application, the code is virtually identical to that of the C/C++ version:

```
hEnumContext := CeFindFirstDatabase(DBType);
if hEnumContext = INVALID_HANDLE_VALUE then
begin
  ShowMessage('Error retrieving DB Info');
  Exit;
end;
for i := 1 to TVCEDB.Items.Count-1 do
  if Assigned(TVCEDB.Items[i].Data) then
    Dispose(TVCEDB.Items[i].Data);
TVCEDB.Items.Clear;
Node := TVCEDB.Items.Add(nil, 'Device Databases');
Node.ImageIndex := 0;
Node.SelectedIndex := 0;
ObjID := CeFindNextDatabase(hEnumContext);
while ((ObjID <> 0) and (ObjID <> ERROR_NO_MORE_ITEMS) and (ObjID <>
ERROR_INVALID_PARAMETER)) do
begin
  CeOidGetInfo(ObjID, CeOIDInfo);
  if CeOIDInfo.wObjType <> OBJTYPE_DATABASE then
  begin
    ObjID := CeFindNextDatabase(hEnumContext);
    Continue;
  end;
  Application.ProcessMessages;
```

The only difference comes when you actually add the items to your TreeView, in that you're adding some data along with the actual text:

```
  with Node, CeOIDInfo.infDatabase do
  begin
    Node := TVCEDB.Items.AddChild(TVCEDB.Items[0], String(szDbase-
Name));
    Data := new(PCeOIContainerStruct);
    TCeOIContainerStruct(Data^).OID := ObjID;
    TCeOIContainerStruct(Data^).OIDInfo := CeOIDInfo;
    ImageIndex := 1;
    SelectedIndex := 3;
```

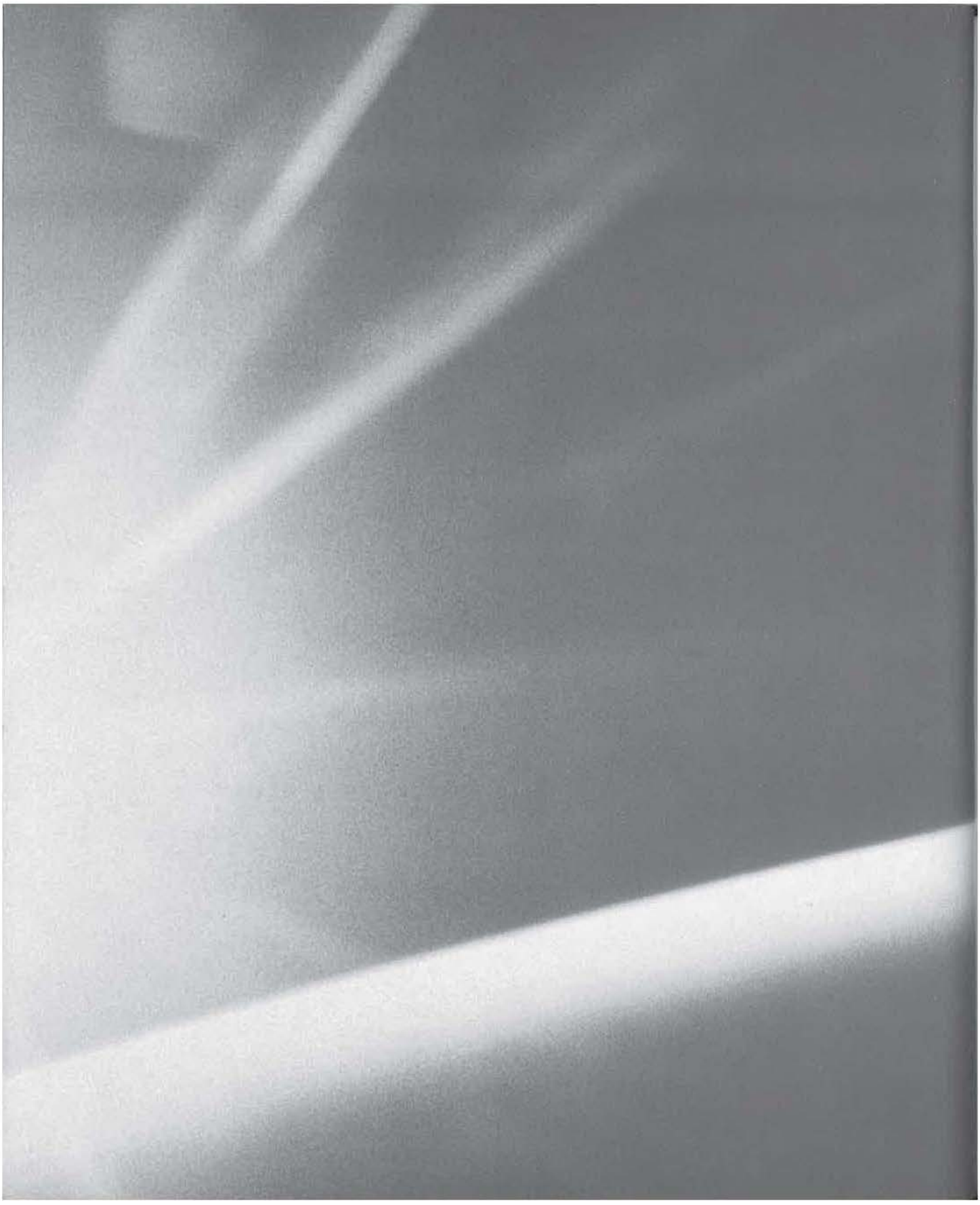
```
    end;
    ObjID := CeFindNextDatabase(hEnumContext);
  end;
  if ObjID = ERROR_INVALID_PARAMETER then
    ShowMessage('An Error occured while retrieving information from the
CE device.');
  TVCEDB.Items[0].Expand(True);
```

With that one simple exception, the two sets of source code are virtually the same.

By using some free source code, you can give your Delphi applications access to the entire RAPI library. And, since Inprise (formerly Borland) does not appear to have plans for a Delphi that compiles a true CE-based executable, this is definitely the next best thing.

Summary

RAPI helps extend the CE application into the Desktop by giving the other machines in your system access to the data and files on the CE device. This is especially important given that the data on these devices wouldn't be worth very much if you couldn't get that data circulated to other machines and other pieces of software. In this chapter, you saw how you could get a Desktop program to perform the very same types of data access possible on the CE devices themselves. In addition, you saw how it was possible to extend even non-Microsoft development products so that they, too, could access and talk to a CE device.



CHAPTER

ELEVEN

11

How CE Talks to the Outside World

- Serial Communications Issues
- Modem-Based Communications Issues
- PC Card/PCMCIA Communications Issues
- Winsock Commmunications Issues

In the last chapter, you saw how Desktop programs can talk to a CE device. In this chapter, we'll be looking at how to get your CE device talking to the outside world, including all of your other devices, Desktop machines, or any other piece of equipment you might have in mind. We'll explore all areas of CE-based communication from serial I/O to CE's Winsock support, starting with a brief overview of your options when it comes to CE communications. Then we'll look at each of the options in detail.

What's in the Box?

There are two aspects to Windows CE-based communication:

- The hardware aspect
- The software aspect

Although this may seem like an obvious distinction, it's not. That's because under Windows 98/NT, an application could open a serial port or begin a Winsock operation without paying much attention to the underlying hardware. Windows CE is closer to the hardware level, however, so you generally have no such luxury. And, as you've seen in previous chapters, each manufacturer may expose different features.

The Hardware Aspect

When it comes to hardware, there are only two ports you can count on:

- A serial port
- An Infrared (IR) port

In addition to these ports, there is an entire set of uncertain hardware, including:

- Modems
- PCMCIA cards

The Serial Port

The default serial port is usually the same one used to connect to the Desktop computer, and it is usually COM1. For the most part, it behaves like a serial port on a Desktop machine. Just as in Windows 98/NT, you can open the port in CE with a call to `CreateFile()`, read from it with `ReadFile()`, and so on. Most of your existing Windows-based serial communications code should port to CE rather easily.

The IR Port

The IR port is more of a gray area, however. Although all commercial CE devices offer an IR port, the port number (i.e., COM1, COM2, etc.) changes from one device to the next.

Also, some manufacturers configure their devices so that both the serial port and the IR port appear as COM1. In this case, the only way to specify which one you want is to open the port and then try to set it into IR mode.

And, as if it isn't confusing enough already, some devices allow you to open a serial port *and* an IR port at the same time. This goes against everything we've ever been told about CE allowing only one serial connection at a time.

Modems and PCMCIA Cards

As for devices that may or may not be available to you, some devices, such as the HP Jornada, have a built-in modem; other devices offer it as an upgrade; and some don't offer a modem at all. Similarly, the PCCard (PCMCIA) slot opens the device up to such hardware as networking cards, cell phone modems, additional serial ports, or higher-speed modems.

TIP If your application calls for additional serial ports, your best bet is either the serial I/O card or the dual serial I/O card offered by Socket Communications of Newark, California (`http://www.socketcom.com`). Both are PCMCIA cards that instantly add one or two RS-232 ports to your CE device.

All of this hardware variety only makes it more of a challenge to design reliable communications-related software.

The Software Aspect

Obviously, with so much possible hardware available on a CE device, there's some complexity to the software as well. When it comes to communications, Windows CE supports a mix of everything from Win32 serial communications functions to Winsock to special blends of Winsock and IR.

Serial Communications

Serial communications haven't changed very much from Windows 98/NT to Windows CE. CE supports 16 of the 23 communications-related API calls, and the ones that aren't there probably won't affect your applications much, if at all. The seven unsupported functions are

- `BuildCommDCB()`
- `BuildCommDCBAndTimeouts()`
- `CommConfigDialog()`
- `GetCommConfig()`
- `GetDefaultCommConfig()`
- `SetCommConfig()`
- `SetDefaultCommConfig()`

It is possible to do almost everything these functions do with the functions that are supported by Windows CE. For instance, most of the functionality of `GetCommConfig()` can easily be replaced with the function `GetCommState()`, which is supported by CE.

The following is a list of the 16 functions that are supported under Windows CE:

- `ClearCommBreak()`
- `ClearCommError()`
- `EscapeCommFunction()`
- `GetCommMask()`
- `GetCommModemStatus()`
- `GetCommProperties()`

- `GetCommState()`
- `GetCommTimeouts()`
- `PurgeComm()`
- `SetCommBreak()`
- `SetCommMask()`
- `SetCommState()`
- `SetCommTimeouts()`
- `SetupComm()`
- `TransmitCommChar()`
- `WaitCommEvent()`

With this many supported functions, it's clear that the few functions that are missing shouldn't affect you too much.

NOTE For more information on some of the missing API functions, see Chapter 3.

IR Communications

IR communications on a CE device are available in three classes:

- Raw IR
- IrCOMM
- Infrared Sockets (IrSock)

Raw IR *Raw IR* means that you'll be using the IR port as though it was any other serial port. There is no special handshaking and no error handling.

The most difficult part about using the IR port as a raw serial port comes from the fact that, as noted above, every manufacturer seems to assign a different port number to the IR port. With some of the earlier CE devices, it was possible to query a certain key in the registry to determine the IR port number, but it seems that all manufacturers do not store that information in the same place.

The only truly reliable way to determine the logical designation of the IR port is to loop through all of the ports and ask each one if it's the IR port.

The way to ask a port whether it is an IR port or a standard serial port is to call `EscapeCommFunction()` and pass in the handle to the open port and a flag (SETIR) indicating that you want to put the port into IR mode:

```
EscapeCommFunction(hPort, SETIR);
```

Since this function only returns TRUE if the port is an IR port, it's safe to assume that if the call fails, the port is not an IR port, and you need to keep looking. The most efficient and effective way to use this trick is probably to create a routine to detect the presence of the IR port. To do this, first initialize the result of the function to 0 to indicate that an IR port could not be found and then allocate memory to store the "name" (COM1, COM2, etc.) of the comm port:

```
result = 0; //zero indicates error, could not find port
szPort = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
```

Then, loop from 1—the lowest comm port number—to MAX_PORTS, a predefined constant indicating the highest port number to test:

```
for (i=1; i <= MAX_PORTS; i++)
{
```

WARNING

The MAX_PORTS value differs greatly from one device to another. On some devices, such as the Casio E-10, the highest comm port number is COM3; on the Casio 2400, the highest comm port number is COM4; and on the HP Jornada, the highest comm port number is COM6!

Next, construct the name of the comm port using the value of `i` as the number of the comm port:

```
_tcscpy(szPort, TEXT("COM"));
_itow(i, szNum, 10); //convert i to string
_tcscat(szPort, szNum);
_tcscat(szPort, TEXT(":"));
```

NOTE

In the above code, there is a call to the RTL function `_itow()`, which only works with Unicode-based strings. Ideally, the function to use is `_itot()`, which works with the generic-string types. However, `_itot()` is not supported on all of the CE platforms. See Appendix A for more information.

WARNING You'll notice that the last step in the above code is to append a colon to the port name. Although Windows 98 and Windows NT do not require that the port name be followed with a colon, Windows CE does.

The next step is to open the port whose name you've just created. Just as you would on Windows 98/NT, do this using the `CreateFile()` function:

```
hPort = CreateFile(szPort, GENERIC_READ | GENERIC_WRITE,
  0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if (hPort != INVALID_HANDLE_VALUE)
{
```

Now, if you've been able to successfully open the port, you can apply the trick of attempting to set the port to IR mode. If it returns TRUE, that means you found the IR port:

```
  if (EscapeCommFunction(hPort, SETIR))
  {
    //Ir Port Found!!!
    result = i;
    CloseHandle(hPort);
    break;
  }
}
```

The rest of the function is just clean up and remembering to return the numeric value representing the IR comm port:

```
  CloseHandle(hPort);
  }
  LocalFree(szPort);
  return result;
}
```

When fully assembled (and with some function calls to display the status of your search in a list box), the full function looks like this:

```
int DetectIRPort(void)
{
  int i, result;
  TCHAR * szPort;
  TCHAR szNum[4];
  HANDLE hPort;
```

```
    result = 0; //zero indicates error, could not find port
    szPort = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
    for (i=1; i <= MAX_PORTS; i++)
    {
      _tcscpy(szPort, TEXT("COM"));
      _itow(i, szNum, 10); //convert i to string
      _tcscat(szPort, szNum);
      _tcscat(szPort, TEXT(":"));
      SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Now testing
        port:"));
      SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)szPort);
      SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("..."));
      hPort = CreateFile(szPort, GENERIC_READ | GENERIC_WRITE,
        0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
      if (hPort != INVALID_HANDLE_VALUE)
      {
        if (EscapeCommFunction(hPort, SETIR))
        {
          SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("IR Port
            Found!!!!!!!!!!!!!!!!!!"));
          SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)szPort);
          result = i;
          CloseHandle(hPort);
          break;
        }
      }
      CloseHandle(hPort);
      }
      LocalFree(szPort);
      return result;
    }
```

When integrated into a sample test application, the result looks like the application shown in Figure 11.1.

IrCOMM In addition to the raw IR, which we've just examined, there is another kind of IR communication available under Windows CE, IrCOMM. This is a more reliable form of IR serial communication that provides handshaking, error handling, and many other features that make it an attractive alternative to raw IR.

FIGURE 11.1:
The IrDetect application

The best part about all of these additional features is that they're completely transparent—as far as your code is concerned, the IrCOMM port is just like any other serial port. In other words, the IrCOMM port can be used as though it were a standard serial port. That means you can call `CreateFile()`, `ReadFile()`, etc., while still enjoying the benefits of a more reliable connection.

In fact, the only difference between raw IR and IrCOMM that you'll actually notice is that the IrCOMM port is on a different logical comm port number than the raw IR port! To ensure that you open the correct IrCOMM port, you'll need to consult the registry. The key containing the information about the IrCOMM port is `HKEY_ LOCAL_MACHINE\Drivers\BuiltIn\IrCOMM`. The actual port number is contained in the value `Index` under this key. Or, as it would appear in code:

```
HKEY hKeyIR;
if (ERROR_SUCCESS == RegOpenKeyEx (HKEY_LOCAL_MACHINE,
TEXT("Drivers\\BuiltIn\\IrCOMM"), 0, KEY_READ, &hKeyIR))
{
      DWORD dwIRport;
      DWORD dwSizeData = sizeof (DWORD);
      if (ERROR_SUCCESS == RegQueryValueEx (hKeyIR, TEXT("Index"),
         NULL, NULL, (LPBYTE)&dwIRport, &dwSizeData))
      {
          WCHAR wszMsg[64];
```

```
        wsprintf (wszMsg, TEXT"IrCOMM port number is: %u"),
          dwIRport);
        MessageBox (NULL, wszMsg, TEXT("IR Port Info"), MB_OK);
}
```

Infrared Sockets (IrSock) Infrared Sockets is a Winsock-like wrapper for communicating over the Infrared port. Refer to the "Winsock-Based Communications" section later in this chapter to see what it takes to convert a simple Winsock-based application to an Infrared Sockets application.

Modem-Based Communications

There are really three classes of modems when it comes to CE:

- Standard external modems connected via the serial port
- Built-in modems
- PC Card/PCMCIA modems

Standard External Modems Standard external modems are just that: standard. There is nothing special about opening a serial port and dialing a modem connected to that serial port when working with Windows 98/NT, and that's true under Windows CE as well.

Built-in Modems It's tempting to conclude that built-in modems work just like external modems—and they do. The trick, however, is finding the internal modem, or more correctly, finding the comm port of the internal modem. The way to do this is to iterate through the `HKEY_LOCAL_MACHINE\ExtModems` key of the registry. Under this key, there will be a key for each of the modems the device knows about. For example, all CE devices have a default Hayes Compatible modem setting that refers to any external modem on COM1. Therefore, under the `HKEY_LOCAL_MACHINE\ExtModems` key, there is a `Hayes Compatible` key. To get the actual comm port name, which you can then use to open the port, simply retrieve the string stored in the Port value of the `Hayes Compatible` key. Or, in code, open the `ExtModems` key under `HKEY_LOCAL_MACHINE` as the first step:

```
RegOpenKeyEx (HKEY_LOCAL_MACHINE, TEXT("ExtModems"), 0, KEY_READ,
  &hKey);
if (hKey)
{
```

Next, create a `while` loop that retrieves the names of the subkeys under the `ExtModems` key (i.e., the subkeys for each of the different modems).

```
retCode = ERROR_SUCCESS;
i = 0;
while (retCode == ERROR_SUCCESS)
{
  cbName = MAX_PATH;
  memset(szSubKeyName, 0, MAX_PATH);
  retCode = RegEnumKeyEx(hKey, i, szKeyName,&cbName, NULL, NULL,
NULL, NULL);
  i++;
```

If you are able to successfully enumerate the key, then try to open the key so you can retrieve the values you're after:

```
if (retCode == (DWORD)ERROR_SUCCESS)
{
  memset(szModemInfo, 0, MAX_PATH);
  memset(szPortInfo, 0, MAX_PATH);
  memset(szDevName, 0, MAX_PATH);
 RegOpenKeyEx (hKey, szKeyName, 0, KEY_READ, &hSubKey);
```

If you are able to open the key, you can retrieve the `Port` value and the `FriendlyName` of the modem:

```
if (hSubKey)
{
  dwBytes = MAX_PATH;
  RegQueryValueEx (hSubKey, TEXT("Port"), NULL, &dwType,
    (LPBYTE)szPortInfo, &dwBytes);
  dwBytes = MAX_PATH;
  RegQueryValueExW(hSubKey, TEXT("FriendlyName"), NULL, &dwType,
    (LPBYTE)szDevName, &dwBytes);
```

Optionally, you could then format the two strings retrieved and add the formatted string to a ComboBox:

```
wsprintf (szModemInfo, TEXT("%s %s"), szPortInfo, szDevName);
SendDlgItemMessage(hwnd, IDC_CBOMODEMS, CB_ADDSTRING, 0,
  (LPARAM)szModemInfo);
```

Finally, clear out your strings for the run through the next subkey:

```
memset(szModemInfo, 0, MAX_PATH);
memset(szPortInfo, 0, MAX_PATH);
```

```
            memset(szDevName, 0, MAX_PATH);
            RegCloseKey(hSubKey);
          }
        }
      }
    }
```

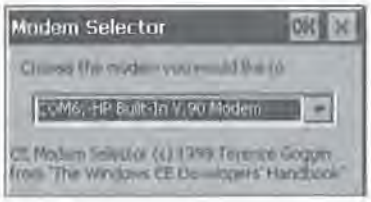
If you then create a simple dialog with a ComboBox and put this code into, say, a WM_INITDIALOG message handler, the final assembled message handler looks like this:

```
case WM_INITDIALOG:
{
    HKEY  hKey, hSubKey;
    DWORD dwDisposition;
    DWORD dwType;
    DWORD dwBytes = 0;
    TCHAR *szSubKeyName;
    TCHAR *szKeyName;
    TCHAR ClassName[MAX_PATH] = TEXT(""); // Buffer for class name.
    DWORD cbName;
    TCHAR *szPortInfo;
    TCHAR *szDevName;
    TCHAR *szModemInfo;
    DWORD retCode;
    int i;

    szSubKeyName = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
    szKeyName  = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
    szPortInfo  = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
    szDevName  = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
    szModemInfo  = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);

    RegOpenKeyEx (HKEY_LOCAL_MACHINE, TEXT("ExtModems"), 0, KEY_READ,
      &hKey);
    if (hKey)
    {
      retCode = ERROR_SUCCESS;
      i = 0;
      while (retCode == ERROR_SUCCESS)
      {
        cbName = MAX_PATH;
        memset(szSubKeyName, 0, MAX_PATH);
```

```
            retCode = RegEnumKeyEx(hKey, i, szKeyName,&cbName, NULL,
              NULL, NULL, NULL);

            i++;
            if (retCode == (DWORD)ERROR_SUCCESS)
            {
              memset(szModemInfo, 0, MAX_PATH);
              memset(szPortInfo, 0, MAX_PATH);
              memset(szDevName, 0, MAX_PATH);
              RegOpenKeyEx (hKey, szKeyName, 0, KEY_READ, &hSubKey);
              if (hSubKey)
              {
                dwBytes = MAX_PATH;
                RegQueryValueEx (hSubKey, TEXT("Port"), NULL, &dwType,
                  (LPBYTE)szPortInfo, &dwBytes);
                dwBytes = MAX_PATH;
                RegQueryValueExW(hSubKey, TEXT("FriendlyName"), NULL,
                  &dwType, (LPBYTE)szDevName, &dwBytes);
                wsprintf (szModemInfo, TEXT("%s %s"), szPortInfo,
                  szDevName);
                SendDlgItemMessage(hwnd, IDC_CBOMODEMS, CB_ADDSTRING, 0,
                  (LPARAM)szModemInfo);
                memset(szModemInfo, 0, MAX_PATH);
                memset(szPortInfo, 0, MAX_PATH);
                memset(szDevName, 0, MAX_PATH);
                RegCloseKey(hSubKey);
              }
            }
          }
        }
        LocalFree(szSubKeyName);
        LocalFree(szKeyName);
        LocalFree(szPortInfo);
        LocalFree(szDevName);
        LocalFree(szModemInfo);
        return TRUE;
    }
```

The resulting Modem Selector dialog looks something like the one pictured in Figure 11.2.

FIGURE 11.2:
The Modem Selector dialog



PC Cards/PCMCIA Modems PCMCIA modems are really just another kind of PCMCIA device. In the next section, we'll be looking at all PCMCIA devices as a group, regardless of whether the actual card is a modem, a serial I/O card, or a network interface card.

PC Cards/PCMCIA Cards and Communications

PC cards/PCMCIA cards are fairly transparent when it comes to communications. Your application could open a comm port and never really know that the port it just opened was really, say, a PCMCIA modem.

However, there *are* two occasions when your application will care about the PCMCIA slot:

- When the application starts and needs to find out if the desired card/port is inserted into the slot and available for use
- When the application is running and the card in the slot changes (i.e., a card is added or removed)

When the Application Starts Officially, the Microsoft documentation says that when your application starts, you can use a function called `EnumPnpIds()` to retrieve a double-NULL terminated list of strings representing the device(s) currently inserted in the PCMCIA slot(s). However, there is one problem with this function: it doesn't exist in any of the `.h` files for any versions of Windows CE!

TIP The Pnp in `EnumPnpIds()` stands for *Plug and Play*.

This doesn't mean you can't get a list of available PCMCIA cards when your program starts, however; you just can't get that list using the `EnumPnpIds()`

function. Instead, you can create a function that does the exact same thing by querying the registry.

In this case, the key you're interested in is HKEY_LOCAL_MACHINE\Drivers\Active. In this key you will find a set of double-digit subkeys numbered 00 to nn, where nn is a double-digit integer.

NOTE

Each of these double-digit subkeys specifies a different driver or hardware component that is currently being used by the device. The values of nn are assigned somewhat sequentially, with the OEM system components taking the lower numbers first. For instance, on many systems, the 00 entry contains information about the sound component, WAV1:. The value of nn assigned to a PCMCIA card depends on the number of times any cards have been inserted since the device was last reset. If the OEM has used, say 00 through 09 for system devices, the first PCMCIA card inserted will get an nn value of 10, the second card a value of 11, and so on. Also, if a card is inserted, removed, and then inserted at a later time (but before a reset) that card may or may not be assigned a different value than the one it had the last time it was used.

The way this set of double-digit subkeys helps make it possible to get information about the PCMCIA cards currently available on the device is that all PCMCIA-related subkeys will have a value called PnpId. If an entry has this value and a Name value containing the string COM, then it is a serial communications card of some kind.

NOTE

Note that this method also detects compact flash cards, so if your application expects a comm port in the form of a compact flash card, this technique will work to detect its presence as well.

Using the same logic employed above in the modem detection routine, you can start by opening the HKEY_LOCAL_MACHINE\Drivers\Active key:

```
RegOpenKeyEx (HKEY_LOCAL_MACHINE, TEXT("Drivers\\Active"), 0, KEY_READ,
  &hKey);
```

NOTE

We present only a portion of the full code here so as to avoid duplicating the modem detection code above. The full code for the PnpId enumeration is on the CD for this book, in the directory for this chapter.

Just as before, you can enumerate the nn subkeys:

```
RegEnumKeyEx(hKey, i, szKeyName,&cbName, NULL, NULL, NULL, NULL);
```

And for each subkey enumerated, you can attempt to retrieve the PnpId and the device name (i.e., COM1, COM2, etc.):

```
//...
RegQueryValueEx (hSubKey, TEXT("PnpId"), NULL, &dwType,
  (LPBYTE)szPnpId, &dwBytes);
//...
RegQueryValueExW(hSubKey, TEXT("Name"), NULL, &dwType,
  (LPBYTE)szDevName, &dwBytes);
```

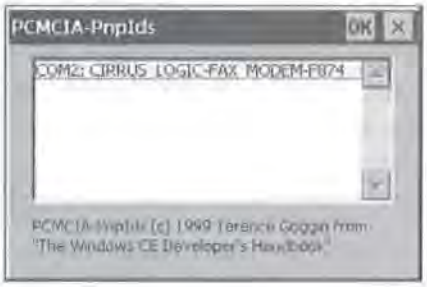
If the string retrieved from the PnpId value is non-NULL, add this item to your list box of PCMCIA devices:

```
if (_tcscmp(szPnpId, TEXT("")) != 0) //there was a PnpId
{
    wsprintf (szDevName, TEXT("%s %s"), szDevName, szPnpId);
    SendDlgItemMessage(hwnd, IDC_LSTPNPIDS, LB_ADDSTRING, 0,
      (LPARAM)szDevName);
}
```

When this code is hooked up to a dialog box with a ListBox on it, the result is something like Figure 11.3.

FIGURE 11.3:

PCMCIA-PnpIds application



TIP

It turns out that the EnumPnpIds()actually does exist, it's just completely undocumented. John Psuik, technical editor of this book, has included a sample on the CD documenting how to call this function, if you prefer to go the undocumented route.

When the Card in the Slot Changes If you are doing serial communications in your application, you're going to have to know when the card in the PCMCIA slot changes.

Why? Well, first you have the issue of error prevention or detection. If your application is in the middle of a file transfer over a PCMCIA-based serial port, and the user pulls the card out, your application will immediately know an error has occurred. However, your application will be much more robust if it is able to determine that the source of the error was the user removing the card and not, say, a problem with the other computer.

Second, even if your application is not actually using a comm port at the moment a card is removed or added, you'll probably want to update the comm port options that you offer the user. If the user inserted a modem or a serial I/O card, it's a safe guess they want to use that port with your program, and they don't want to have to restart your application in order to refresh the list of available comm ports.

Now that we know the advantages of monitoring changes in the PCMCIA slot, let's investigate how to do it. Like the techniques outlined in the section above, detecting a change in the current PCMCIA card requires some undocumented—or at least under-documented—CE tricks. Here again, the documentation says that an application need only respond to the `WM_DEVICECHANGE` message in order to be notified when a PCMCIA card is inserted or removed. However, there appears to be a small problem with this in that other portions of the documentation indicate that `WM_DEVICECHANGE` doesn't even exist on CE!

So what's the truth?

The `WM_DEVICECHANGE` message *does* exist, but it's not defined in the `winuser.h` header file where all of the other `WM_` messages are defined. Instead, the `WM_DEVICECHANGE` message and a number of related constants and structures are all defined in a separate header file called `dbt.h`.

The first step, then, in handling the `WM_DEVICECHANGE` message is including this header file.

The second step is correctly interpreting the `wParam` value that the `WM_DEVICECHANGE` message passes to your application. From testing, it appears that there are only two values that matter under Windows CE:

- `DBT_DEVICEARRIVAL` A card has just been inserted into the slot.
- `DBT_DEVICEREMOVECOMPLETE` A card has just been removed from the slot.

Code to handle the wParam of the message, then, might look like this:

```
switch(wParam)
{
case DBT_DEVICEARRIVAL:
  //Card was inserted
  break;
case DBT_DEVICEREMOVECOMPLETE:
  //Card was removed
  break;
}
```

The lParam of WM_DEVICECHANGE is a pointer to a structure that should tell you a little bit of information about the device. The trick to using the lParam is that it points to one of several possible structures, depending on what type of card has been inserted. In order to determine what type of card and, therefore, which structure lParam is pointing to, you must first cast the lParam to a generic structure and read one of that structure's members.

The generic structure is called DEV_BROADCAST_HEADER and is defined as follows:

```
struct _DEV_BROADCAST_HEADER
{
  DWORD   dbcd_size;
  DWORD   dbcd_devicetype;
  DWORD   dbcd_reserved;
};
```

In order to determine the type of card that was just inserted or removed, you must examine the value of the structure's dbcd_devicetype member. The possible values for dbcd_devicetype are

- DBT_DEVTYP_OEM Unspecified OEM type card
- DBT_DEVTYP_PORT Serial or parallel port
- DBT_DEVTYP_NET Network resource

NOTE There are other possible values for the dbcd_devicetype member, but they do not appear to have meaning under Windows CE.

Once you've tested the `dbcd_devicetype` value, you can cast the `lParam` to a more detailed structure specific to the card's type. For some reason, however, it appears that CE reports all PCMCIA cards as being `DBT_DEVTYP_PORT` cards—in other words, it considers all cards to be serial- or parallel-port cards.

The positive side of this is that you only have to worry about casting the `lParam` to one type of structure; the negative side is that you'll have to work even harder to differentiate the comm port cards from the other types of cards.

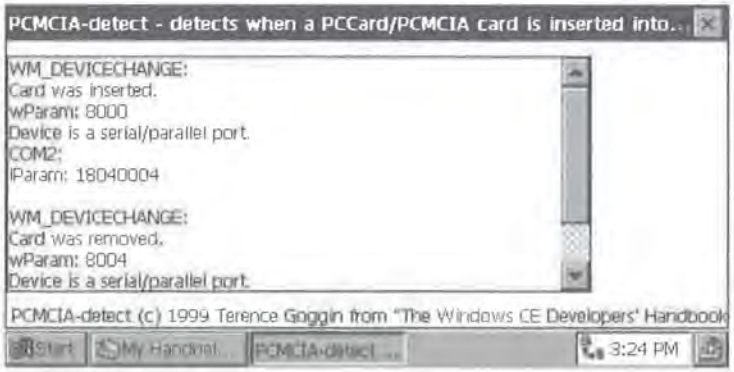
The comm port–specific structure is called `DEV_BROADCAST_PORT` and is defined as follows:

```
typedef struct _DEV_BROADCAST_PORT_W
{
  DWORD    dbcp_size;
  DWORD    dbcp_devicetype;
  DWORD    dbcp_reserved;
  wchar_t    dbcp_name[1];
};
```

As you may have guessed, the member of this structure that actually makes it possible to determine whether or not the card in question is a comm port is `dbcp_name[1]`. If the card is a comm port, the name will contain the text `"COM"` followed by a number and a colon. The good news about this is that the name is all you need in order to call `CreateFile()`and open that port.

If you now put everything that you know about PCMCIA cards into a simple testing utility, you'll wind up with something like the application pictured in Figure 11.4.

FIGURE 11.4:
The PCMCIA Detect application



Of course, the WM_DEVICECHANGE message handler does the real work of this application:

```
case WM_DEVICECHANGE:
 TCHAR * szParam;
 TCHAR szNum[MAX_PATH];
 PDEV_BROADCAST_HDR pdbhDeviceHeader; //the generic card structure
```

First, add a status message to a list box to let the user know that notification was received:

```
SendMessage(hListBox, LB_ADDSTRING, 0,
  (LPARAM)TEXT("WM_DEVICECHANGE:"));
```

Next, based on the value of wParam, alert the user as to whether the card is being inserted or removed:

```
switch(wParam)
{
case DBT_DEVICEARRIVAL:
  SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Card was
    inserted."));
  break;
case DBT_DEVICEREMOVECOMPLETE:
  SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Card was
    removed."));
  break;
}
```

Next, you can convert wParam and lParam to strings and display their actual values:

```
szParam = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
_tcscpy(szParam, TEXT("wParam: "));
_ltow(wParam, szNum, 16); //convert i to hex string
_tcscat(szParam, szNum);
SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)szParam);
memset(szParam, 0, MAX_PATH);
_tcscpy(szParam, TEXT("lParam: "));
_ltow(lParam, szNum, 16); //convert i to hex string
_tcscat(szParam, szNum);
```

Now, treat the lParam as the generic device information structure:

```
pdbhDeviceHeader = (PDEV_BROADCAST_HDR)lParam;
```

Then, attempt to determine the type of card:

```
switch(pdbhDeviceHeader->dbch_devicetype)
{
```

If it's a serial or parallel port, go ahead and treat the `lParam` as a specific `DEV_BROADCAST_PORT` structure:

```
case DBT_DEVTYP_PORT:
  PDEV_BROADCAST_PORT pdbpPortDeviceHeader;
  pdbpPortDeviceHeader = (PDEV_BROADCAST_PORT)lParam;
```

You can also let the user know the type of card, and, from the `DEV_BROADCAST_PORT` structure, the name of the port, as well:

```
  SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Device is a
    serial/parallel port."));
  SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)pdbpPortDevice
    Header->dbcp_name);
  break;
```

As noted before, all PCMCIA cards appear to be recognized by CE as serial or parallel ports. Therefore, the default clause here is mostly for the sake of good coding:

```
default:
  SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Device is of
    an unknown type."));
}
```

When fully assembled, the entire message handler looks like this:

```
case WM_DEVICECHANGE:
  TCHAR * szParam;
  TCHAR szNum[MAX_PATH];
  PDEV_BROADCAST_HDR pdbhDeviceHeader;
  SendMessage(hListBox, LB_ADDSTRING, 0,
    (LPARAM)TEXT("WM_DEVICECHANGE:"));
  switch(wParam)
  {
  case DBT_DEVICEARRIVAL:
    SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Card was
      inserted."));
    break;
  case DBT_DEVICEREMOVECOMPLETE:
```

```
    SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Card was
      removed."));
    break;
  }
  szParam = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
  _tcscpy(szParam, TEXT("wParam: "));
  _ltow(wParam, szNum, 16); //convert i to hex string
  _tcscat(szParam, szNum);
  SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)szParam);
  memset(szParam, 0, MAX_PATH);
  _tcscpy(szParam, TEXT("lParam: "));
  _ltow(lParam, szNum, 16); //convert i to hex string
  _tcscat(szParam, szNum);
  pdbhDeviceHeader = (PDEV_BROADCAST_HDR)lParam;
  switch(pdbhDeviceHeader->dbch_devicetype)
  {
  case DBT_DEVTYP_PORT:
    PDEV_BROADCAST_PORT pdbpPortDeviceHeader;
    pdbpPortDeviceHeader = (PDEV_BROADCAST_PORT)lParam;
    SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Device is a
      serial/parallel port."));
    SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)pdbpPortDevice
      Header->dbcp_name);
    break;
  case DBT_DEVTYP_NET:
    SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Device is a
      network interface card."));
    break;
  default:
    SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT("Device is of
      an unknown type."));
  }
  SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)szParam);
  LocalFree(szParam);
  SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)TEXT(""));
  break;
```

You now have a complete set of PCMCIA detection routines.

TIP This WM_DEVICECHANGE code should also work quite nicely under Windows 98/NT!

Winsock-Based Communications

Windows CE supports a subset of the Winsock 1.1 specifications. What this means on a practical level, though, is that most of your Windows 98/NT Winsock-based code will not port to Windows CE. If, however, you have some Windows 3.*x* code that you want to port, that process will probably go very smoothly.

Of course, there's still a lot you can do with Winsock 1.1. In fact, you can really do just about everything you can do with Winsock 2, but your code will look very different. For instance, under Winsock 1.1, there is no `WSAAsyncSelect()` function to help you process socket-related events. In fact, there are only five `WSA` functions supported under Windows CE:

- `WSACleanup()`
- `WSAGetLastError()`
- `WSAIoctl()`
- `WSASetLastError()`
- `WSAStartup()`

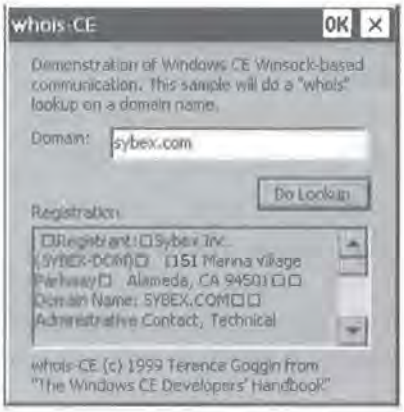
All of the `WSA`–related functionality is only accessible through Berkeley Sockets–style functions, such as `accept()`, `bind()`, and `receive()`.

Winsock 1.1 To learn how to make the Winsock 1.1 functions actually do something, let's create a simple *whois* client. As you know, *whois* is a very simple protocol used to retrieve registration information about domain names. All the client does is connect to the server, send the request in the form `whois somedomain.com`, and wait for the result. The server will return some text containing the desired information or a message indicating that the domain is available.

First, create a simple dialog, with an Edit control, a multiline Edit control, and a button, so that when you're done, you have a dialog looking something like the one in Figure 11.5.

The user will be able to enter a domain name to look up in the single-line Edit control and view the results in the multiline Edit control. The actual look-up code will be triggered whenever the user clicks the button.

FIGURE 11.5:
The whois program



The first step to take in the WM_COMMAND event handler for the button when coding your whois client is to initialize Winsock with the WSAStartup() function, just as you would under Winsock 2:

```
if (WSAStartup(wVersionRequested, &wsaData))
{
    WSACleanup();
    return TRUE;
}
```

Next, create the client socket with a call to the socket() function:

```
sockWhois = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sockWhois == INVALID_SOCKET)
{
    WSACleanup();
}
```

Then, populate a SOCKADDR_IN structure with information about the whois server's address and port number to connect on:

```
saddrInternic.sin_family = AF_INET;
//198.41.0.6 - whois.internic.net *the* whois server
saddrInternic.sin_addr.S_un.S_un_b.s_b1 = 198;
saddrInternic.sin_addr.S_un.S_un_b.s_b2 = 41;
saddrInternic.sin_addr.S_un.S_un_b.s_b3 = 0;
saddrInternic.sin_addr.S_un.S_un_b.s_b4 = 6;
saddrInternic.sin_port = htons(43); //whois port number
```

Now, attempt to connect to the server:

```
if (connect(sockWhois, (LPSOCKADDR)&saddrInternic,
  sizeof(SOCKADDR_IN)) == SOCKET_ERROR)
{
    closesocket(sockWhois);
    WSACleanup();
}
```

Next, construct the string that will be your request to the server by concatenating "whois " with the domain name the user entered into the edit box:

```
strcpy(szWhois, "whois ");
GetDlgItemText(hwnd, IDC_DOMAIN, szUnicodeDomain, MAX_PATH);
WideCharToMultiByte(CP_ACP, 0, szUnicodeDomain, wcslen(szUnicode-
  Domain) * sizeof(TCHAR), szTemp, MAX_PATH, NULL, NULL);
strcat(szWhois, szTemp);
```

WARNING Note the use of the `WideCharToMultiByte()` function above. Although CE is Unicode-based, all of the Internet protocols still use ANSI text strings. Therefore, you must convert the Unicode-string of the edit box to an ANSI string. For more information on working with Unicode, see Chapter 2.

You can now send this request string to the whois server:

```
if (send(sockWhois, szWhois, strlen(szWhois), 0) == SOCKET_ERROR)
{
    closesocket(sockWhois);
    WSACleanup();
}
```

Loop until you receive the full text of the reply from the server, calling the `recv()` function to get a small portion of the text each time through the loop:

```
do
{
    iResult = recv(sockWhois, szTemp, sizeof(szTemp), 0);
    if (iResult == SOCKET_ERROR)
        break;
    strcat(szWhois, szTemp);
    memset(szTemp, 0, MAX_PATH);
}while (iResult != 0);
```

Now you can close the socket and clean up:

```
closesocket(sockWhois);
WSACleanup();
```

When some code to display the errors to the user is added and the entire routine is assembled, it looks like this:

```
case IDC_BTNLOOKUP:
  {
    WORD wVersionRequested = MAKEWORD(1,1);
    WSADATA wsaData;
    SOCKET    sockWhois;
    int iResult;
    SOCKADDR_IN saddrInternic;
    char *szWhois;
    char *szTemp;
    TCHAR *szUnicodeDomain;
    DWORD dwNumWritten;
    SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("Initializing
      Winsock."));
    if (WSAStartup(wVersionRequested, &wsaData))
    {
        SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("ERR: Unable
          to initialize Winsock."));
        WSACleanup();
        return TRUE;
    }
    sockWhois = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sockWhois == INVALID_SOCKET)
    {
        SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("ERR: Unable
          to create client socket."));
        WSACleanup();
        return TRUE;
    }
    SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("Socket
      created."));
    saddrInternic.sin_port = htons(43); //whois port
    saddrInternic.sin_family = AF_INET;
    //198.41.0.6 - whois.internic.net *the* whois server
    saddrInternic.sin_addr.S_un.S_un_b.s_b1 = 198;
    saddrInternic.sin_addr.S_un.S_un_b.s_b2 = 41;
```

```
saddrInternic.sin_addr.S_un.S_un_b.s_b3 = 0;
saddrInternic.sin_addr.S_un.S_un_b.s_b4 = 6;
if (connect(sockWhois, (LPSOCKADDR)&saddrInternic, sizeof(SOCKADDR
  _IN)) == SOCKET_ERROR)
{
    SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("ERR: Unable
      to connect to server."));
    closesocket(sockWhois);
    WSACleanup();
    return TRUE;
}
SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("Connected to
  server."));
szWhois = (char *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH * 2);
szUnicodeDomain = (TCHAR *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH * 2 *
  sizeof(TCHAR));
szTemp = (char *)LocalAlloc(LMEM_ZEROINIT, MAX_PATH);
strcpy(szWhois, "whois ");
GetDlgItemText(hwnd, IDC_DOMAIN, szUnicodeDomain, MAX_PATH);
WideCharToMultiByte(CP_ACP, 0, szUnicodeDomain, wcslen(szUnicodeDo-
  main) * sizeof(TCHAR), szTemp, MAX_PATH, NULL, NULL);
strcat(szWhois, szTemp);
SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("Sending
  request."));
if (send(sockWhois, szWhois, strlen(szWhois), 0) == SOCKET_ERROR)
{
    SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("ERR: Unable
      to send request to server."));
    closesocket(sockWhois);
    WSACleanup();
    return TRUE;
}
memset(szWhois, 0, MAX_PATH * 2);
strcpy(szWhois, " ");
memset(szTemp, 0, MAX_PATH);
SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)TEXT("Receiving
  reply."));
do
{
    iResult = recv(sockWhois, szTemp, sizeof(szTemp), 0);
    if (iResult == SOCKET_ERROR)
        break;
```

```
            strcat(szWhois, szTemp);
            memset(szTemp, 0, MAX_PATH);
        }while (iResult != 0);
        closesocket(sockWhois);
        WSACleanup();
        MultiByteToWideChar(CP_ACP, 0, szWhois, strlen(szWhois), szUnicode-
          Domain, MAX_PATH * 2 * sizeof(TCHAR));
        SendMessage(hEdtInfo, WM_SETTEXT, 0, (LPARAM)szUnicodeDomain);
        WriteFile(hFile, szUnicodeDomain, _tcslen(szUnicodeDomain) *
          sizeof(TCHAR), &dwNumWritten, NULL);
        LocalFree(szWhois);
        LocalFree(szUnicodeDomain);
        LocalFree(szTemp);
    }
```

When you actually look at this code, you can see that Winsock 1.1, while crude compared to Winsock 2, is not too difficult to use and should be adequate for most of your communications needs.

Infrared Sockets (IrSock) Infrared Sockets is version of Winsock 1.1 that uses the IR port as a transport medium. There really isn't a whole lot of difference between IrSock and Winsock—in fact, just make sure you take these precautions when converting your Winsock code to IrSock code:

- Always include the `af_irda.h` header file.
- Use `AF_IRDA` instead of `AF_INET` when creating a socket.
- Use the `SOCKADDR_IRDA` structure instead of the `SOCKADDR_IN` structure.

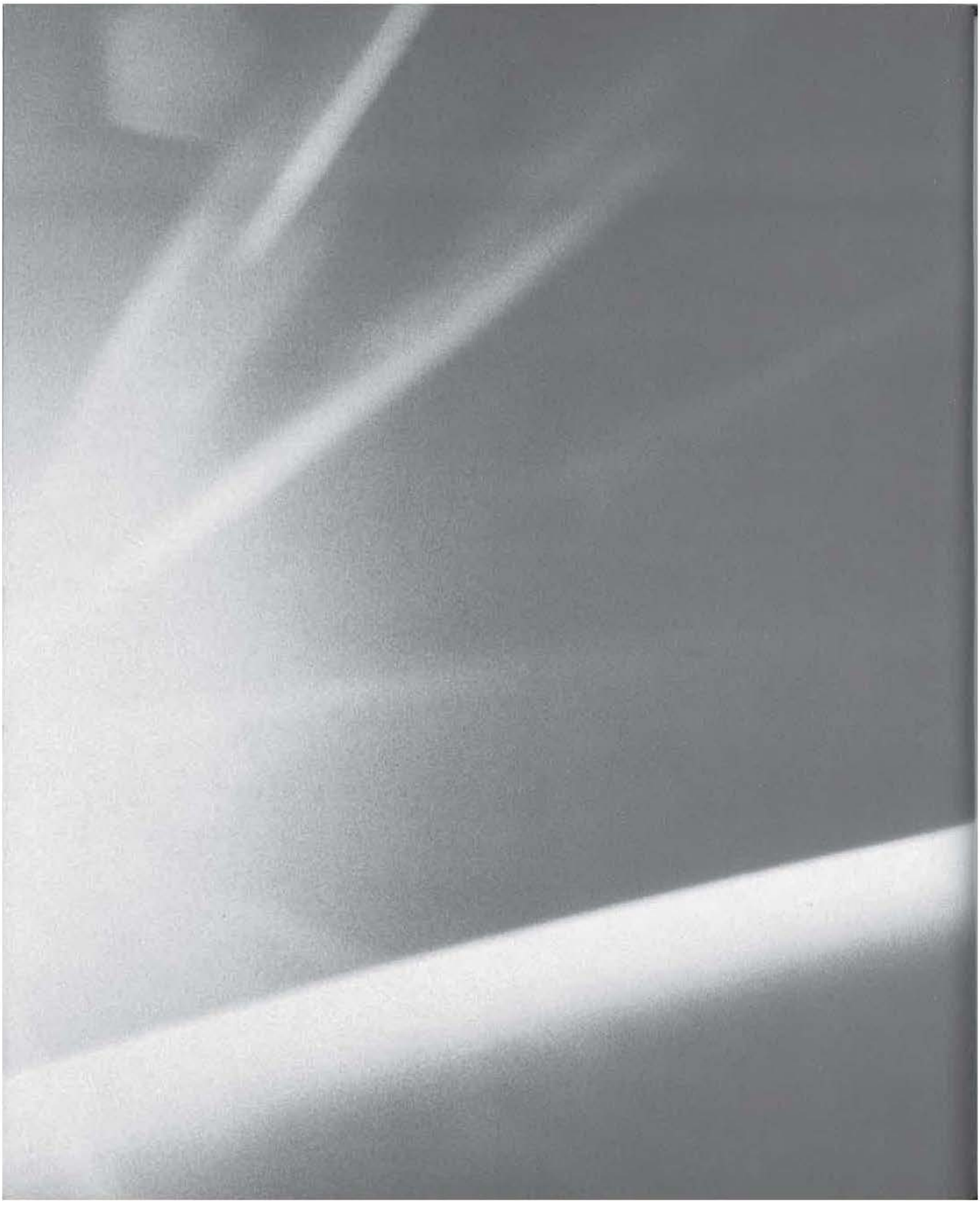
With these three exceptions, *everything* else is exactly the same as standard Winsock operations.

TIP

MFC's `CCeSocket` does provide some rudimentary support for asynchronous notification; you may want to consider using MFC if code that you are porting depends heavily on asynchronous notification.

Summary

In this chapter, you've explored CE's communications hardware and software sides, and you saw how to smoothly integrate the two in order to make our applications as robust and as flexible as possible. You started with simple serial communications and moved up to Infrared ports. You learned how to detect internal modems on a CE device and you even delved into a bit of undocumented Windows CE in order to detect changes in the PCMCIA hardware. Finally, you took a look at Winsock 1.1 and IrSock technologies.



CHAPTER

TWELVE

12

Third-Party Database Engines

- Raima's RDM/CE
- Sybase's Adaptive Server Anywhere
- Oracle Lite
- Comparison of Features

In Part I of this book, you learned about the database engine that Windows CE provides as part of the API. As you saw, CE's database engine has some quirks and limitations that may affect its usefulness, depending on your purpose. To summarize, the real strengths of the CE database engine are:

- Zero cost: it's part of the CE operating system/API.
- Automatic Data Compression: all CE databases are compressed by default.
- Simple searches: simple searches of the indexed field can be performed.
- Simple sorts: simple sorts through sort orders can be performed.

However, the CE database engine falls a little short in these areas:

- Unusual table structures/organization
- Complex API calls
- No data relationships
- No support for SQL

NOTE In addition to the above list, there are a number of reports on the CE newsgroups and mailing lists that bog down the CE database engine since it's dealing with such large amounts of data.

In this chapter, we'll address the issue of what to do when the CE database engine doesn't meet your needs. Specifically, we'll be looking at some of the third-party database engines that are available for Windows CE.

When it comes to data access on CE, there are three leading third-party vendors:

- Raima offers Raima Data Manager for CE
- Sybase offers Adaptive Server Anywhere for CE
- Oracle offers Oracle Lite for CE

A little experimentation is necessary to ensure that you choose the best database engine for your needs. Therefore, each of these three vendors has prepared a sample application as well as some introductory text about their database engine.

These three sections will comprise the bulk of this chapter. After that, we'll do a feature-by-feature comparison of the engines so that you can pick the one that best meets your development needs.

Raima's RDM/CE for Data Storage

When you need to store more data access it in ways that are difficult using CE's native facilities, it's time to look at a more robust data manager. You can write your own, or use a third-party product. In this section, we examine one of those third-party products, Raima Data Manager for Windows CE (RDM/CE).

RDM/CE is a network model database with extensive indexing and navigation capabilities. Its data files are binary compatible with RDM for Win32, Win16, and even DOS, so you can prepare data on whichever platform is most convenient to you.

Network and Relational Data Models

While Windows CE offers incredible computing speed for the price, current platforms are limited in the amount of data they can store. On some platforms, main memory is shared with mass storage and can be as small as 2MB. If your application requires a lot of data storage, finding the most compact data format can make your application a real winner. As with any tool for CE development, you'll want to consider the files size of the DLL involved. More important in database management, though, is the size of the data files themselves as you add your data.

The most commonly used data storage format in business computing is the relational model. Relational technology uses common fields and indexes to allow navigation from one record to another. For example, consider an invoicing system in which you have CUSTOMER, INVOICE and INVOICE_ITEM tables. Each CUSTOMER will have a unique CUST_ID, and the CUST_ID will likely be indexed. Each INVOICE will have its INV_ID, which will also be indexed, along with the INV_CUST_ID stored in the INVOICE table. Similarly, each INVOICE_ITEM record will contain an indexed field, the II_INV_ID.

Navigating from a specific customer record involves first retrieving the CUSTOMER record. This gives you the CUST_ID. You then must look in the INV_CUST_ID index for the INVOICE table, retrieving each INVOICE record

in turn, as you find it in the index. Retrieving all the INVOICE_ITEM records involves the II_INV_ID index, since the INV_ID is the key to retrieving the data.

This technology is easy to understand, practical, and flexible. But in a constrained environment like Windows CE, how can you cut some corners, save some space, and still get the data retrieval you need? The network model can offer a real advantage.

Rather than relying on a common field and performing index lookups to do a JOIN, the network model offers direct record-to-record navigation, either manually or using what RDM calls *sets*. In an RDM set, one record is designated an owner, and one or more record types serve as members.

In the example above, the CUSTOMER record would be the owner of the INVOICE record, and the INVOICE record would in turn be the owner of the INV_ITEM record. Embedded in each CUSTOMER record are the physical addresses of the first and last INVOICE records. Each INVOICE record contains the physical address of the related CUSTOMER record and the physical addresses of the next and previous INVOICE records for the same customer.

Navigating the data is more direct, since reading the CUSTOMER record immediately yields the physical address of the first INVOICE record. Reading the INVOICE record instantly yields the address of the first INV_ITEM record.

In RDM, this physical address is called the database address, or DB_ADDR. Sets can be managed using a number of set functions, or DB_ADDR fields can be stored in your own programs during processing. They can even be added to the other fields in your database definition.

This direct record-to-record navigation, with its advantage of speed and optional elimination of indexes, makes network databases like RDM an excellent match for Windows CE.

Introducing HpcLadr

A ladder puzzle is an old word game. An example of a ladder puzzle is, given a start word of CAP and a destination word of PIN, you can transform one letter at a time thus: CAP, CAN, PAN, PIN. HpcLadr is a Windows CE application that makes it possible to create these puzzles yourself; it also allows you to solve the puzzles it creates. Figure 12.1 shows the completed HpcLadr running on a CE device.

FIGURE 12.1:
HpcLadr running on a CE device

The first item you need so you can write a program like HpcLadr is a database of words. You can prepare this database with an application running on Windows NT, using RDM for Windows NT. This program, called LadrPrep, is also in the \LADDER directory. When you run it, you will find it has only two actions it can perform:

- Import words
- Export words

HpcLadr comes complete with a set of data and index files; you will only need to use LadrPrep if you wish to change the words in the HpcLadr database.

The database that LadrPrep creates is structurally simple. Each word is entered into a table of similarly sized words. The database is searched for other words that could be adjacent in a ladder puzzle, and mutual DB_ADDR pointers are added. For example, CAP points to CAN, and CAN points to CAP.

In designing the database, you must take into account how you intend to use it. In this case, you know that your only use for the database is to find a specific word and from it find all ladder-adjacent words. Therefore, instead of using RDM's full set capabilities, it makes sense to use a very simple database entry for each word, such as this record definition for three-letter words:

```
record word3 { char w3[3]; DB_ADDR adjacent; }
```

In the adjacent field, store the physical address of the first related word you find. The size of the entire record is only 7 bytes. However, rather than pointing to the actual word, the DB_ADDR points to a junction record (only 8 bytes and no index necessary):

```
record junction { DB_ADDR target_word; DB_ADDR next_adjacent; }
```

The logic of adding the word CAP to a database already containing CAN could be represented with pseudocode that looks something like this:

```
Add new word to database; its DB_ADDR is db_new
Search database for adjacent words
For each adjacent word db_adj
     Create two new junction records, j1 and j2
     Set j1's next_adjacent to db_adj's adjacent
     Set j1's target_word to db_adj
     Set db_adj's adjacent to j1
     Set j2's next_adjacent to db_new's adjacent
     Set j2's target_word to db_new
     Set db_new's adjacent to j2
End for
```

The ability to directly store physical database addresses here gives you both speed and compactness of data representation. Using relational technology, the junction set has to contain not two 32-bit quantities, but one column for the target word and one column for the adjacent word. Even if you limit yourself to seven-letter words, a 14-byte record is required instead of an 8-byte record. In addition, for practical use the junction table must be indexed, adding overhead of at least 11 bytes per record, for a total of 25 bytes instead of RDM's 8 bytes.

This may seem like a contrived example, but think about the data in your own application: how much data access travels along predictable paths, parent record to child record? Could you speed up your application *and* reduce storage requirements by trying network technology?

Adding a list of words to LadrPrep creates a database in which every word is directly linked to all words that differ by only one letter. It will also contain a number of words, like xylophone, which aren't linked to any others, because no other word differs from them by only one letter. Therefore, the LadrPrep function's Export Word List only exports words that have links.

If you decide to create your own database for HpcLadr using LadrPrep, you will get the smallest database by performing the following steps:

1. Import the word list.
2. Export the word list.
3. Import the word list once more.

This will ensure that the database is as compact as possible.

Once the database is prepared, writing HpcLadr is pretty easy. In its Make a Ladder mode, HpcLadr simply displays a list from which you may choose words. Once you choose a step in your ladder, HpcLadr shows all the adjacent words, eliminating any that are closer to the beginning of the ladder than the current step. This way, you are guaranteed that any puzzle you construct using HpcLadr will use the shortest possible route between your chosen words, given the vocabulary used by the program.

In its Solve a Ladder mode, HpcLadr shows the starting word, the ending word, and a varying number of hint words. You fill in the ladder by typing in the adjacent words or by clicking on the words in the hint list.

Implementing both modes is quite easy using RDM and the database you designed. Given a specific word, retrieve its record and read the junction record's DB_ADDR. Each junction record contains a DB_ADDR `target_word`. Retrieve that and add a word to the list of words displayed as adjacent to your original word. Navigate to the junction record's `next_adjacent`, and you're at a new junction record. Simple, direct, fast, and there's no need for indexes.

Putting RDM to Work

You can find the file `RaimaDM.zip` in the `\Raima` directory on this book's companion CD. It contains two files: `README.txt` and `RDMCE.zip`. Please read the license agreement in `README.txt`; if you agree with the terms, you are welcome to try the full RDM/CE product, encrypted with a key disclosed in the license agreement. More information is available direct from Raima at `http://www.raima.com`.

Sybase's Adaptive Server Anywhere

As part of their SQL Anywhere Studio, Sybase offers Adaptive Server Anywhere (ASA) for Windows CE. ASA builds on Sybase's technology to deliver a small-footprint, self-tuning, fully functional RDBMS engine for applications operating in remote or mobile computing environments.

In the next section, we'll be looking at a sample application provided by Sybase for the purpose of demonstrating some of the features and benefits of ASA for Windows CE.

Sybase's Adaptive Server Anywhere Sample Application

This sample application demonstrates database access, multi-user features/user rights, and report generation using SQL stored procedures. The application allows a user to log in to the database. Based on their user rights, they will be able to choose a report to view. The reports are stored in the database as stored procedures. When the user selects a report to view, the ASA database engine will execute a stored procedure. The application will then display the results of this procedure.

TIP Stored procedures are complex SQL statements, stored and managed by the database engine, which can then be executed and referenced like other SQL functions.

A major advantage of storing the reports as stored procedures is that very little SQL code needs to be hard-coded into the application. For example, if a new report needs to be created, it can simply be added to the database without having to recompile and test the application. Similarly, through the replication and synching features of ASA, these new reports can be added to a central database and then populated to all of the CE devices that connect to that central database. Database replication can also be very useful in cases where the data is changing on a consistent basis.

NOTE This sample application does not include database synching functionality, but it is a very simple task to include it.

The Database Schema

The application uses two types of tables:

- Fake sales data
- Report data

For our purposes, the table that's of interest is the Report table. The Report table is organized as follows:

report_id A unique integer column used as the table's primary key.

report_name The "friendly" name of the report. The application uses this column to display the list of available reports.

proc_name The name of the SQL stored procedure that generates the report.

security_level A numeric value used to determine the user's rights.

Administering and Managing ASA for CE Databases

Due to the limited resources of Windows CE, Sybase does not provide database administration tools that run on CE devices. Therefore, you have two options for administering these databases:

- Active Sync
- File copy

Administering the database with Active Sync might be easy for users who are familiar with the Active Sync process. However, as you know, transferring data over the serial port is incredibly slow. Therefore, this option can be fairly time consuming, especially if there are a large number of records that need to be synched.

The second alternative is to copy the database (`.db`) and transaction log (`.log`) from the CE device to your desktop. This, of course, can be done with an Ethernet connection or any other way you prefer to get your CE device connected to your desktop. This method is almost always quicker than synching over a serial connection.

TIP

Since the databases are completely binary compatible, you can start this database on your desktop and administer it directly from Sybase Central. Once the changes are complete, you can recopy it down to your CE device.

Configuring the Application

To set up the demo, follow these steps:

1. Install ASA for CE on the device.
2. Once the setup has been completed, copy both the application and the database files to the root directory of the CE device.
3. Start up the database server by running the following command:

```
"\program files\asa\dbsrv6" -x "TCPIP" -z -n demo asademo.db
```

After the database has started, you are ready to start the application.

ASA for CE Command Line Options

In this example, you're starting the ASA server with several command line options that allow you to tweak the server's operation for your individual needs. For instance, the `-x "TCPIP"` option assumes that the device is connected to the network and that you'll be connecting to it via TCP/IP. If you're not connected to the desktop, simply use `-x ""` instead.

Similarly, the `-z` option gives you debugging information and is not required. The `-n` option assigns the name `demo` to the current session of the server. The `asademo.db` tells the server to start the `asademo.db` database.

Incidentally, Sybase recommends creating a shortcut specifying your most common configuration to avoid typing the full command line each time.

NOTE The default database administrator userid is `dba` and the password is `sql`.

The Demo

In the previous section, you started the database server. Now you'll start the application by launching `asademo.exe`. Figure 12.2 shows the login window. When prompted, you'll enter a userid of **clark** and a password of **sales**.

FIGURE 12.2:
The Login dialog

When you click OK, the application executes the Logon method of the CDemoDB class:

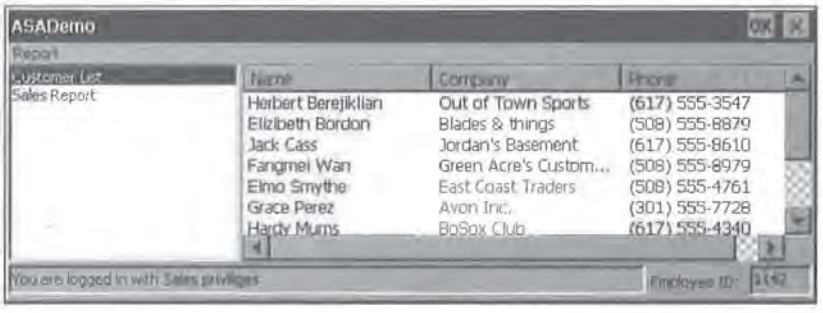
```
result = m_db.Logon(m_user, m_password);
```

Next, the application will attempt to log in to the database. If unsuccessful, it will return an error message. If successful, the application will display the Report dialog, as shown in Figure 12.3.

FIGURE 12.3:
The Report dialog

The server returns the employee ID in the bottom right corner of the page, as well as the user rights assigned to this user in the bottom left corner. Based on these user rights, the application displays all of the reports that this user can access. If you click on Customer List in the `CListBox` labeled Reports List, the application calls the appropriate stored procedure, which is executed by the ASA Server. The application then displays the results of the stored procedure in the `CListView` control. This is shown in Figure 12.4.

FIGURE 12.4:
Results of running the Customer List report



The code performing this operation first retrieves the name of the report from the CListBox control (called m_reports):

```
TCHAR Buffer[60];
m_reports.GetText( m_reports.GetCurSel(), Buffer );
```

Next, translate the friendly name of the selected report into the name of the stored procedure, again using your CDemoDB class:

```
m_db.GetReportProcedure(Buffer, m_empid);  //m_db is of class
CDemoDB
```

Then, after setting up the CListView control, add one record at a time to the CListView, looping until there are no more records to display:

```
for (;;)
{
     if (m_db.GetNextReportRow() == false)
       break;
     FillListCtrl(m_db.m_result1,m_db.m_result2,m_db.m_result3);
}
```

If you now click on Sales Report, the Server will run the Sales Report stored procedure. Once again, the appropriate stored procedure is called, which brings back all of the sales this person has made. As before, the resulting set of data is displayed in the ListView control.

If you then close the application and start it again, this time logging in as a different user, you will see the user rights features at work. This time, log in with a user ID of kelly and a password of manager.

Once you're logged in, you'll see the same dialog as before. However, this time the employee ID in the bottom right corner is different, and the privileges assigned to this user are different than those of the previous user. For instance, this user has rights to view three reports, as opposed to the previous user who could only view two reports. It stands to reason that the results of their reports will also differ.

In fact, that is exactly what happens when you click on Customer List. This time, the stored procedure returns a list of all the customers. The reason the full list is displayed is that this user has privileges to view all customers, whereas the previous user did not. These user privileges are a major advantage of using stored procedures over standard SQL calls.

In conclusion, ASA for CE offers all of the same functionality you've come to expect from a high-performance database engine with all the compactness of a CE application. Keep in mind that the above application shows off only a small portion of the features and benefits offered by Adaptive Server Anywhere for CE.

Oracle Lite Introduction

Oracle Lite is a powerful object-relational client database for mobile enterprise applications. Its built-in replication engine enables data to be synchronized with Oracle 7.3 (or higher) data servers. Its small footprint allows it to run on laptops and PDAs, including Windows CE devices.

Oracle Lite is an enabling technology that allows large-scale deployment of Windows CE devices for enterprise applications such as sales force automation, maintenance crew support, outpatient care support, etc.

Although widely used for personal information management, PDAs have not become mainstream business tools yet for two main reasons:

- The cost of accessing corporate data from mobile devices is prohibitive.
- PDAs could not previously do complex data management and processing.

Prohibitive Cost

Traditionally, the only available technology for accessing corporate data from PDAs has been wireless distant database querying and file exchanges. Wireless communication has two obvious handicaps, however: high cost and slow communication. These handicaps normally make permanent wireless connection of large fleets of PDAs with corporate databases and file systems too expensive.

Lack of Processing Power

Until now, there was no true object-relational database on the Windows CE platform that seriously impaired any development of data-intensive enterprise applications. Oracle Lite changes this.

Oracle Lite solves these two problems and enables large-scale deployment of Windows CE devices for business applications by doing the following:

- It allows fast and inexpensive access to corporate data on mobile devices.
- It enables data-intensive applications on PDAs.

Fast and Inexpensive Access to Corporate Data on Mobile Devices

Thanks to replication technology, a subset of the larger corporate database populates the local database. After the local database is populated, the mobile application accesses and modifies this local subset of data. This allows much faster transactions than wireless database queries. At regular intervals, the local database is synchronized with the master database. During synchronization, only the changes, not the entire subsets, are transferred between the central database and the local databases; the process is very bandwidth-efficient. Overall bandwidth cost is a fraction of that of wireless distant queries, with the same end user benefits.

Data-Intensive Applications on PDAs

Oracle Lite is a true object-relational database on the Windows CE platform and may be accessed through standard methods, such as ODBC. Because Oracle Lite is ODBC compliant, programming is truly easy and uses standard SQL statements. Oracle Lite is the database of choice for building any mobile enterprise application on the Windows CE platform and enables a wide range of mobile applications to be developed. These applications reduce the cost of tasks performed by mobile workers.

For further information regarding Oracle Lite solutions in your industry, visit the Oracle Lite Industry Solutions Guide at `www.oracle.com/mobile/olite/`. To download the Oracle Lite database engine, visit `www.oracle.com/mobile/`.

Oracle Lite for Windows CE

The Oracle Lite database engine is available on all Windows CE Handheld PC (HPC) and tablet devices running Windows CE 2 (and higher). With Oracle Lite, enterprise applications using standard SQL can run on Windows CE and have their data automatically synchronized with enterprise servers. Like Oracle Lite on

the desktop or laptop, Oracle Lite on Windows CE offers a standard ODBC interface and takes full advantage of Oracle Advanced Replication.

Key Features

The key features of Oracle Lite are:

- Fully object-relational Oracle database for Windows CE
- Small footprint: 350KB core engine, 750KB with ODBC support
- Built-in bidirectional replication to Oracle data servers over multiple protocols
- Built-in subsetting and conflict resolution to support multiple unique clients
- Support for wireless client applications

ODBC Support

With the Oracle Lite ODBC driver for Windows CE, porting an existing SQL-based application can be as simple as recompiling it for the target Handheld PC using a tool such as Visual C++. Alternatively, an application can be developed from the ground up that supports complex SQL queries to be executed on the Handheld device. Oracle Lite for Windows CE leverages your investment in the tools and applications you already have by providing open data access. In addition, tools such as Oracle Navigator and SQL*Plus can be used from the desktop to access the Oracle Lite database on Windows CE through the Remote ODBC feature. Using this feature with Oracle Navigator, you can even drag and drop a table from a mainframe to Windows CE or from an Oracle server to Oracle Lite.

Application Development Support

Applications for the Handheld PC are most often developed and tested first on the desktop, after which they are then downloaded to the actual Handheld device. Oracle Lite for Windows CE provides support for this kind of cross-development environment, and developers have three different models to choose from:

- Applications can be developed using Oracle Lite for 95/98/NT first. Once perfected, the applications can then be recompiled for the target CE device.

- Developers can use the CE emulation mode support on an NT workstation. This allows rapid development and testing cycles without having to download to the CE device after every recompilation.
- Applications can be tested directly on the CE device to ensure proper functionality on the target platform.

In addition, Oracle Lite for Windows CE comes with a download utility to simplify application deployment to the CE device.

TIP

Currently, Oracle Lite on Windows CE only works with Visual C++ for CE. Other development tools will be supported as they become available for Windows CE.

Sample Application

The following code samples illustrate how the Oracle Lite database can be accessed on the CE device. The first segment defines an Execute primitive that allows a SQL statement to be passed and a result set to be returned.

```
// Execute a Statement and Return a Result Set
//
CSQLResult* COLiteDB :: Execute (LPCTSTR pszSQL)
{
  SWORD    len;
  SWORD    nCol;
  HSTMT    stmt;
  CWaitCursor wait;

  if (::SQLAllocStmt (m_dbc, &stmt) == SQL_SUCCESS)
  {
    if (::SQLExecDirect (stmt, (BYTE*)::MakeChar (pszSQL), SQL_NTS)
   == SQL_SUCCESS)
    {
      g_szBuff[0] = 0;
      if (::SQLNumResultCols (stmt, &nCol) == SQL_SUCCESS && nCol > 0)
      {
        CSQLResult* result = new CSQLResult (stmt, TRUE); //N Columns

        for (int i = 1; i <= nCol; i++)
        {
```

```
          if (::SQLDescribeCol (stmt, i, (BYTE*)g_szBuff,
sizeof(g_szBuff),
            &len, NULL, NULL, NULL, NULL) == SQL_SUCCESS)
              result->AddColumn ((const char*)g_szBuff);
        }
        return result;
      }
      ::SQLFreeStmt (stmt, SQL_CLOSE);
      ::SQLExecDirect (stmt, g_pszCommit, SQL_NTS);
    }
    else
    {
      UCHAR state[32];
      SDWORD err;
      UCHAR msg[1024];
      SWORD len;

      ::SQLError (m_env, m_dbc, stmt, state, &err, msg, 1024, &len);
      state[6] = 0;
      wsprintf (g_szBuff, TEXT ("SQLExecDirect: Native=%d
message=%s"), err, MakeWideChar ((char*)state));
      ::SQLFreeStmt (stmt, SQL_CLOSE);
    }
    ::SQLFreeStmt (stmt, SQL_DROP);
  }
  return NULL;
}
```

The code segment below shows how the Execute primitive can be called to create a table and insert rows.

```
//////////////////////////////////////////////////////////////////////////
//////
// CSampleApp initialization

BOOL CSampleApp::InitInstance()
{
  COLiteDB db;
  if (db.Connect())
  {
    db.Execute (_T ("CREATE TABLE TT (COL1 VARCHAR2(40), COL2
    VARCHAR2 (50))"));
    if (*db.GetError())
```

```
      AfxMessageBox (db.GetError());

    db.Execute (_T ("INSERT INTO TT VALUES ('TEST1', 'TEST2')"));
    if (*db.GetError())
      AfxMessageBox (db.GetError());

    db.Execute (_T ("UPDATE TT SET COL2 = 'TESTING ORACLE LITE'
WHERE COL1='TEST1'"));

    if (*db.GetError())
      AfxMessageBox (db.GetError());

    CSQLResult* pres = db.Execute (_T ("SELECT * FROM TT"));
    if (pres != NULL)
    {
      const CRowObj* pobj = pres->Fetch();
      while (pobj)
      {
        CString str = (LPCTSTR)pobj->GetAt (0);
        str += TEXT ("  ");
        str += (LPCTSTR)pobj->GetAt (1);
        AfxMessageBox (str);

        pobj = pres->Fetch();
      }
      delete pres;
    }
  }
    // Since the dialog has been closed, return FALSE so that we
   exit the
    // application, rather than start the application's message
   pump.return FALSE;
}
```

The full code sample can be found on the Oracle Lite 3.5 CD.

Oracle Lite Replication

Oracle Lite enables subquery subsetting for mobile users. Subquery subsetting is a technique that allows the server to identify the exact, unique subset of information that each user needs to see or is authorized to see. For instance, each sales

representative gets only the information relating to his or her accounts. This subsetting capability extends to any tables in one-to-one, many-to-one, many-to-many, or one-to-many relationships between relevant tables. Not only do sales reps receive only the customer records unique to them, that uniqueness extends to any dependent or associated records in other tables such as order numbers, addresses, or other related information.

Subquery subsetting works hand in hand with Oracle Lite's extensive, powerful replication capabilities. Users receive only the records of interest to them; in addition, users can update only the records that are their responsiblity. Oracle Lite's replication functionality allows you to automatically copy information between Oracle Lite and Oracle servers. More than just a copy mechanism, replication takes two tables with the same structure and automatically merges them together at two or more locations, giving distributed users data synchronization with a centralized Oracle7 or Oracle8 server running anywhere in your enterprise.

Oracle Lite for CE offers three replication options:

- Wireless replication using Oracle Mobile Agents
- Internet replication
- File-based replication and disk file replication

Wireless Replication Using Oracle Mobile Agents Oracle Lite allows users to do wireless replication using a store-and-forward method. This replication strategy uses Oracle Mobile Agents to pass changes to a replication agent on the server side and receive updated information from the server. The agent then handles all communication with the master site, applying and receiving changes on behalf of the snapshot site. Changes are then batched and returned to the snapshot site.

Oracle Mobile Agents is described more thoroughly in the "Oracle Mobile Agents" section later in this chapter.

Internet Replication With a connection to the Internet, users can invoke replication and send the changes from the snapshot site to a Web application server through HTTP or MIME protocols. The Web application server then communicates with the master site (behind a firewall) to apply changes, receive changes intended for the snapshot site, and send those changes back over the Internet to the snapshot site. Changes can be sent almost any way the Internet allows, even via e-mail.

Internet replication allows any user who has an Internet connection to perform replication with the master site. It also minimizes the number of connections users have to make. For example, if they connect to the Internet to check mail or to browse, they can also send and receive database changes.

File-Based Replication and Disk File Replication With file-based replication mechanisms, Oracle Lite exposes its replication capabilities so that a developer can execute replication and post the snapshot changes to a file. The developer can then choose to move this file to the master site in whatever manner desired including MAPI, FTP, sneaker net, or flash memory cards. The master site will then apply the changes to the master table and place its changes in a file that the developer can move back to the snapshot site so as to apply the snapshot.

File-based replication gives developers control over how replication changes are moved to the master site. It also provides flexibility in how replication is performed during other types of connections (i.e., mail transfers).

Security

Oracle Lite can be made secure either for local data protection purposes or for transmission purposes. Users or user applications can encrypt an Oracle Lite database with a key as a password, send the database to a master site, and then connect to the encrypted database using the same key.

Oracle Mobile Agents

As an integral part of Oracle's mobile computing strategy, Oracle Mobile Agents provides the connectivity required for mobile computing. This flexible, standards-based solution provides the connectivity for all mobile users, whether LAN based, dial-up, or wireless, on a wide variety of platforms.

Anytime, Anywhere Oracle Mobile Agents is a mobile middleware product, that is, an asynchronous, secure, store-and-forward messaging system that provides the foundation on which to quickly build and deploy mobile applications. Oracle Mobile Agents supports a variety of wireless networks, including packet data networks and dial-up and LAN connections. Its scaleable architecture can add mobility to and enhance the productivity of any application in the enterprise.

Optimized for the Mobile Environment The client/agent/server architecture used by Oracle Mobile Agents minimizes traffic over the wireless link, which is often slower and less reliable than a LAN; this is a key feature for any mobile

application. It minimizes traffic by using a software agent to work on behalf of the mobile client. While traditional client/server operations require the client to remain connected for the duration of a transaction, Oracle Mobile Agents requires the client to remain connected only long enough to submit a request to an agent. With combined data compression and carefully optimized message size, expensive connect time can be kept to a minimum. From a user perspective, application performance over wireless links becomes comparable to a LAN connection.

Wireless Replication with Oracle Lite Oracle Lite exploits the features of Oracle Mobile Agents to provide wireless replication. Replication is the synchronization of data between databases, in this case an Oracle Lite database on a client and a server side Oracle7 or Oracle8 database. Wireless replication provides flexibility to a mobile user to replicate data at any time—for example, right after a large order has been entered into the client database—to ensure that the order-processing commences immediately. Using Oracle Mobile Agents for wireless replication requires no additional programming.

Developing Applications and Agents The Oracle Mobile Agents architecture requires that a client application be modified (or written from scratch) to communicate with the flexible Oracle Mobile Agents' API. Client applications can access Oracle Mobile Agents via OLE v2, or Windows' DLLs, or by using the ActiveX interface. For new applications, all of the typical development packages can be used.

An agent must also be written that will act on behalf of the application at the server, and this agent will also communicate with the Oracle Mobile Agents infrastructure via the same API. The role of the agent is to keep all of the highly interactive network traffic within the corporate network, thus minimizing network round trips over the mobile link. Additionally, the agent allows work to be performed for the client even when the client is not active or connected to the system. Agents are a combination of code you write and a code library, called the Agent Event Manager, supplied as part of Oracle Mobile Agents.

Secure and Reliable The public nature of most wireless services raises valid security concerns. Oracle Mobile Agents addresses these concerns by providing configurable authentication, encryption, and tamper proofing, all of which are implemented using industry standard algorithms. Further, Oracle Mobile Agents' store-and-forward architecture guarantees message delivery. If a mobile worker is unavailable or suddenly loses coverage, messages are queued rather than discarded. Together, these services allow the mobile worker to work when and where they want, without worrying about network availability and knowing that corporate information is safe.

Samples

The sample program in the Odbc directory creates the table `tt` (in the database `\oracle\polite.odb`) and inserts one new row into the table `tt` containing the data:

```
'Test1'|'Testing Oracle Lite'
```

TIP The statements used to create the table and insert data into the table may be seen in the file `SAMPLE.CPP` located in the `Odbc` directory.

The program then prints out the contents of all rows in `tt` in separate dialog boxes on the Windows CE device.

The REPSVR sample under the `replication` subdirectory demonstrates bi-directional replication between the Windows CE device and an Oracle Web Server.

When compiling `REPSVR.exe`, the application is copied to the Windows CE device. The first logon screen requires the server user name, password, connect string, the `<Server URL>\repcartx`, and the Communication Type (HTTP or OMA). A sample URL is: `HTTP://testserver.us.oracle.com/repcartx`. The second logon screen requires the local user name, password, and ODBC DSN (such as `POLITE`).

The main screen gives you the option to create a snapshot, add a snapshot to the snapshot group, drop a snapshot, or replicate.

Comparison of Features

Because the different vendors' sample applications only show off a small portion of each product's full potential, in this section we'll look at some comparison tables that show what each one offers. This way, you can pick the one that best suits your needs. In addition, for the purposes of having a full and complete picture, we'll also include the CE database engine in the tables.

The features we'll be examining are:

- Is the data automatically compressed?
- How does the data get copied to/synched with desktop data?

- How much storage space does the engine require?
- Does it support SQL?
- Does it support stored procedures?
- Does it support user privileges?

Is the Data Automatically Compressed?

CE	Raima	Sybase	Oracle
Yes	No	No	No

CE's own native engine is the only one that automatically compresses data, unless you explicitly tell it not to.

How Does the Data Get Copied To/Synched with Desktop Data?

CE	Raima	Sybase	Oracle
Custom Coding (some automatic functionality)	Custom Coding	Automated for us	Automated for us

How Much Storage Space Does the Engine Require?

CE	Raima	Sybase	Oracle
None (part of the OS)	500K	1MB	1MB

Does the Engine Support SQL?

CE	Raima	Sybase	Oracle
No	Subset	Yes	Yes

Does the Engine Support Stored Procedures?

CE	Raima	Sybase	Oracle
No	No	Yes	Yes

Does the Engine Support User Privileges?

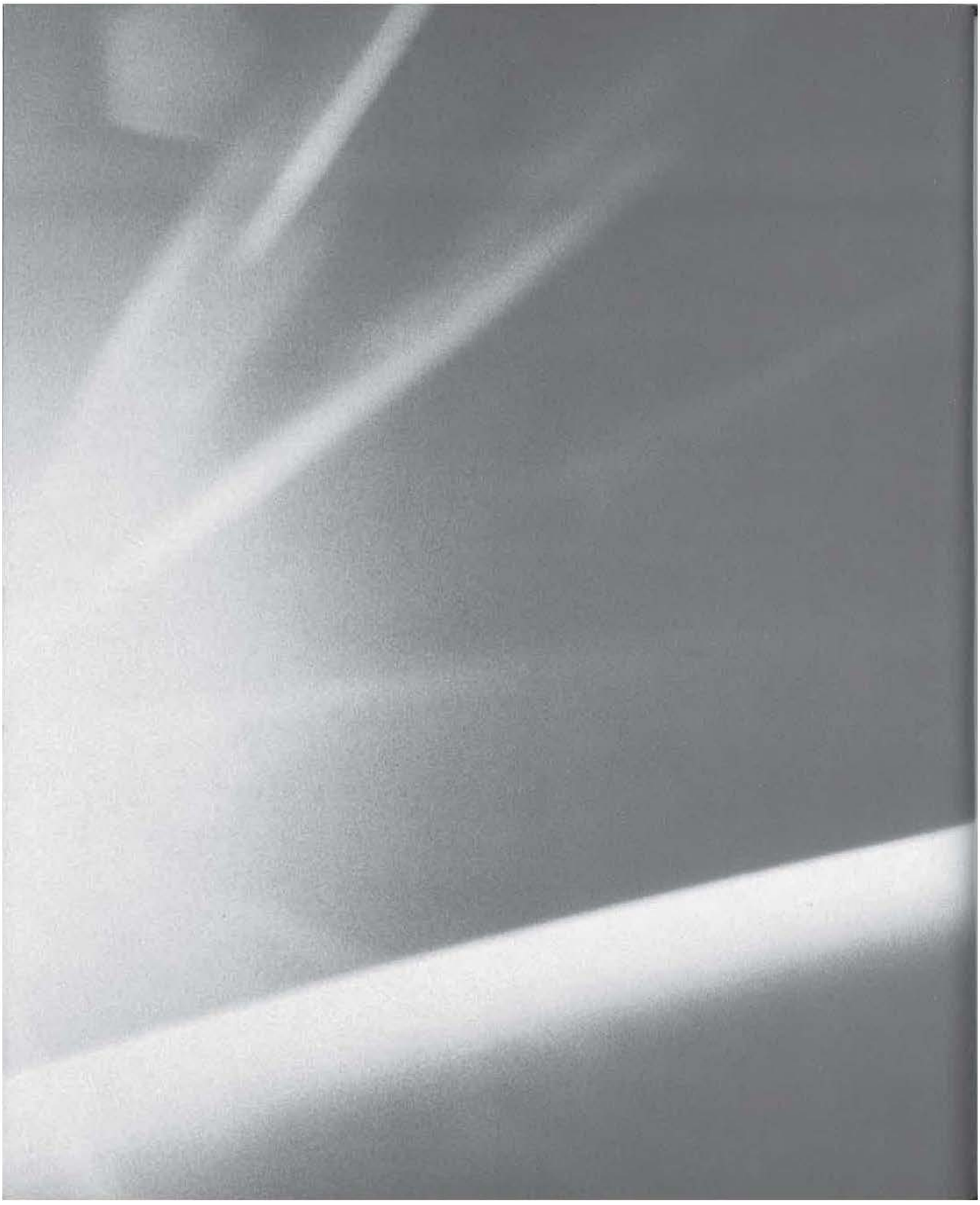
CE	Raima	Sybase	Oracle
No	No	Yes	Yes

Summary

In this chapter, we looked at some of the features offered by third-party databases for Windows CE. We also compared the CE database engine to those of Raima, Sybase, and Oracle, with the following results:

- CE's database engine is towards the low end, but it is still very attractive due to its compression and its zero footprint.
- Raima's RDM/CE is somewhere towards the high-middle range, with a number of professional features, such as true indexing and very small footprint.
- Oracle and Sybase are very high end, with features that we've come to expect from these two database engine leaders.

Just as with any technical implementation issue, there are always tradeoffs. Choosing the database engine that's right for you is no different. For many applications, the CE database engine will be adequate, if not overkill. For others, stored procedures will be a must.



CHAPTER

THIRTEEN

13

Windows CE Case Studies and Cost Analysis

- Study 1: Inventory Management System
- Study 2: Insurance Agents in the Field
- Study 3: Choosing Your Development Machine

In this chapter, we'll look at several example case studies that are based on real case studies and stories about how CE devices are being used in the real world. The purpose of this chapter is to provide some sample ideas on how CE is being used or could be used and the costs of implementing CE-based solutions.

Study 1: Inventory Management System

An inventory management system is perhaps the most frequently cited ideal Windows CE solution. The vision of CE devices as an inventory management tool often goes something like this:

"The user performing the actual inventorying would walk through the warehouse with either a Palm-size PC or Handheld PC and a bar-code scanner attached to this device. They would travel the warehouse and scan in the bar-codes on the boxes as they went. The information could be automatically recorded in a database and then synchronized with the main database when the device is attached to a Desktop computer sometime later."

But is the technical implementation of this process correct? Is it cost-effective for the company? Is this really the best use of some CE devices? In short, does it make sense for the company to do this?

Of course, some of these questions are really a matter of opinion (and some might even cause fierce debate), but others can be answered with a careful analysis of the plan.

The Technical Issues

First, let's examine the technical details. When it comes to the technical details, there are really only three major questions:

- Is the bar-code scanner attachment the most efficient way to implement this solution?
- Is it best to synchronize the data afterward or is there a better alternative?
- Which form factors can be used?

The Bar-Code Scanner

It seems that a bar-code scanner is, in fact, the best way to implement this solution, for a number of reasons. First, bar codes are extremely efficient. In fact, it's safe to say that almost every large inventory management system uses bar codes in some way.

Second, as you saw in Chapter 1, the leading bar-code devices for Windows CE scan the bar code and enter the input into the keyboard buffer as though the numeric values of the bar code were typed in by the user. The advantage of such a system is that you don't have to write any special code to support a bar-code scanner. Simply write your application to accept keyboard input, then hook up the bar-code scanner. This means that you can deliver a fully functional solution, even if the customer/user decides not to include the bar-code scanner in the final package. Further, since these readers are available in either PCMCIA or CF (compact flash) implementations, you still have maximum flexibility in terms of any unused port(s) on the CE device.

NOTE As mentioned in Chapter 1, the maker of these CE barcode readers is Socket Communications (`http://www.socketcom.com`).

Synchronization vs. a Wireless Connection

The next issue to look at is whether the data should be synchronized after it's been collected or whether you should use a wireless connection to ensure that the data is always accurate up to the minute.

The main benefit of synchronization is its low cost of implementation. CE devices are automatically configured to hook up to a Desktop machine and transfer data over a serial cable. The downside to a synchronization-based solution is that your main databases will never be 100 percent accurate and will always lag behind the real count of items in your warehouse by at least eight hours.

A wireless LAN solution, however, ensures that the data in your main databases is at most a few seconds old. But there is also a higher cost associated with this way of updating your main database.

Also, the size of the warehouse becomes an issue here because a wireless LAN hub has a range of approximately 500 square feet. This means that it might well require more than one or two wireless LAN hubs to cover the entire warehouse. While this might appear to be expensive, it may be that the benefit of having all

of the inventory data updated as the information is being scanned in far outweighs the cost of the wireless LAN hubs.

The main difference between this issue and the bar-code issue is that this one must be decided before any application development is begun. If you're going to connect to a database server, you'll need to build the application with support for some kind of wireless client libraries. As you can imagine, this is a much more complicated task once the application's already been written.

Form Factor Choices

As with any part of this system, there are trade-offs when it comes to choosing the right form factor. For inventory management, though, there are really only two practical form factors available:

- Palm-size PC
- Handheld PC

Palm-Size PC The advantages of the Palm-size PC for this application are somewhat obvious: their small size means that the user is free to move about and can access harder-to-reach areas more easily. The disadvantage of a Palm-size PC for this solution is that PPCs only have a single CF (compact flash) slot. This means that there is no way to connect them to a wireless LAN, as the wireless LAN cards are PCMCIA based. And, even if there were wireless LAN cards that worked off of the CF slot, there wouldn't be any place for the bar-code scanner to attach to the device if the CF slot were used for the LAN connection. Clearly, then, the PPC devices are only a good solution in cases where the data does not need to be updated live but can be synched later.

Handheld PC The Handheld PC devices have that extra slot that makes them a more powerful solution to this problem. HPCs have a place for a bar-code scanner (the CF slot) and a place for a wireless LAN connector (the PCMCIA slot). However, the HPC devices are almost always more expensive and somewhat larger in bulk. Low cost is still one advantage possessed by the PPCs.

Cost

Let's now take a look at the costs of implementing such a solution to this inventory management problem. Depending on the system purchased, traditional, non-CE-based inventory management devices can cost upward of $2000 per unit; let's see how that compares to the possible Windows CE–based solutions.

WARNING Please note that all prices should be considered approximate guides only. Prices may—and probably will—change by the time you read this.

Palm-Size PC–Based Solution Cost Analysis

Table 13.1 shows the Palm-size PC–related form factor expenses.

TABLE 13.1: Palm-Size PC–Related Expenses

Item	Maker	Model	Cost
Palm-size PC	Casio	E-11	$299
Bar-code Scanner CF card	Socket Communications	N/A	$180 (est.)
Total Cost			**$479**

Handheld PC–Based Solution Cost Analysis

Table 13.2 shows the Handheld PC–related form factor expenses. The main difference here is that you must figure in the additional costs for the wireless LAN hardware.

TABLE 13.2: Handheld PC–Related Expenses

Item	Maker	Model	Cost
Handheld PC	Casio	PA-2400	$499 (est.)
Bar-code Scanner CF card	Socket Communications	N/A	$180 (est.)
RangeLAN2 PCMCIA card	Proxim	N/A	$595
Total Cost			**$1274**

In addition to these expenses, which are per-device costs, there is the cost of one or more wireless hubs (about one hub per 500–1000 square feet). This cost is displayed in Table 13.3.

TABLE 13.3: Wireless LAN Hub Cost

Item	Maker	Model	Cost
RangeLAN2 Ethernet Access Point	Proxim	N/A	$1895

NOTE As mentioned in Chapter 1, you should be using Proxim's (`http://www.proxim.com`) RangeLAN2 family of products for your wireless LAN calculations.

Although the HPC solution costs more, it also offers much more in the way of network connectivity and, more specifically, data access.

Conclusion of Study 1

This example is a very popular real world use of CE devices. It turns out that in many cases, the benefits of either solution—whether you choose the PPC or the HPC—far outweigh the cost. As mentioned above, most bar-code-based inventory systems cost more than $2000 per unit. At a cost of approximately $500, the PPC-based solution is not only very high-tech, it's also very, very cheap!

Study 2: Insurance Agents in the Field

One issue that seems to have a lot of consumers' attention is the benefits of an HPC/Pro device, such as the Hewlett-Packard Jornada, over a standard Windows 98/NT laptop. One of the advantages that's often cited is the 10-hour (or greater) battery life of an HPC/Pro device as compared with the one-and-a-half to two-hour battery life of a laptop.

Here's a practical example that shows just how much of a difference an HPC/Pro device can make.

Consider insurance agents working on claims in the field. They need to take pictures of property damage and then file those pictures, along with their reports, once they return to the office. Currently, however, they have no really good way to preview their pictures and make sure that the pictures did the job.

Solution

If these agents could use digital cameras that store images on compact flash memory cards, they could transfer the pictures over to the handheld device using the CF slot. Their HPC/Pro devices, using Sierra Imaging's Image Expert for CE, would allow them to preview and, if necessary, retake any pictures that didn't turn out as they expected.

Here again, you are faced with a similar set of questions: Is this the best use of the insurance agency's money? What are they really getting for their money?

Let's examine how well this solution meets the agents' needs.

The Technical Issues

There are really two technical issues that need to be addressed here:

- Is a digital camera the right tool for the job?
- Is a CE-based HPC/Pro the right tool for the job?

Digital Camera

First, let's examine the question of whether or not a digital camera is the right tool for the job.

Many people who rely on some kind of photography in their professions have probably already begun to switch to digital cameras. There are many advantages to digital cameras, such as more pictures, not having to get your pictures developed, being able to preview your pictures, etc.

So a digital camera is probably the right tool for this job on its technical merit alone. As you'll see in the cost analysis section, there's another very compelling reason to go to a digital camera.

HPC/Pro

The HPC/Pro is an obvious choice here simply based on the battery life. If the agents were to use a laptop instead, they would only be able to work with it on battery power for a maximum of 2 hours. The HPC/Pro class devices all run for 10 or more hours on a fully charged battery.

The extra eight hours of battery life means that the agents could easily spend a full eight-hour day in the field without worrying about their computer going down.

Further, there's already very inexpensive image editing/viewing software available for CE devices, so there's no real difference between the CE-based solution and the laptop-based solution in terms of the functionality needed for this purpose.

Cost

Now, let's take a look at cost and see if our technical conclusions are justified. First, you've got to select your hardware.

For the camera, let's use the Canon PowerShot A5, which seems to be a fairly popular model of digital camera. It uses compact flash memory and costs about $699.

And, since we'll be doing a pricing comparison here, let's choose a laptop. Most reviews seem to like comparing the HPC/Pro devices to the Sony Vaio ultrathin laptop, which costs about $1999.

Now let's see how this stacks up against the Windows CE–based solution.

WARNING Again, please note that all prices should be considered approximate guides only. Prices may—and probably will—change by the time you read this.

Laptop-Based Solution Cost Analysis

Table 13.4 shows the laptop-related expenses, if you decide to choose that type of hardware.

TABLE 13.4: Laptop-Related Expenses

Item	Maker	Model	Cost
Laptop	Sony	Vaio	$1999 (est.)
Digital Camera	Canon	PowerShot A5	$699 (est.)
Total Cost			**$2698**

HPC/Pro-Based Solution Cost Analysis

Now let's look at the HPC/Pro-based solution. Table 13.5 shows the Handheld PC–related expenses, if you decide to choose that form factor. The main difference here is that you must figure in the additional costs for the wireless LAN hardware.

TABLE 13.5: HPC/Pro-Related Expenses

Item	Maker	Model	Cost
HPC/Pro	Hewlett-Packard	Jornada	$899 (est.)
Digital Camera	Canon	PowerShot A5	$699 (est.)
Graphics S/W	Sierra Imaging	Image Expert	$50
Total Cost			**$1648**

NOTE Sierra Imaging's (`http://www.sierraimaging.com`) Image Expert for CE is the leading graphics software for Windows CE.

In addition to being the right choice based strictly on technical merit, the Windows CE–based solution is also significantly cheaper—by $1050, to be exact.

Conclusion of Study 2

An HPC/Pro device proved to clearly be the winner on technical merits, as well as the winner on price. This is a perfect example of putting CE to work.

Study 3: Choosing Your Development Machine

One type of study that will probably be of the most interest to you as a developer is how to choose your first CE device for development purposes.

Unless you're committing to CE full time, chances are you'll want to test the waters, so to speak, by purchasing one device and working with it as your development machine until you have had a chance to fully evaluate Windows CE. Yes, you can theoretically develop an entire application in the emulator, but it's not generally recommended practice. At some point, you'll need or want to test your application on a real, honest-to-goodness device, just to make sure that the emulator didn't give you a false impression of how your program was running.

Solution

In this case, there really is no clear-cut solution. But let's take a look at the pros and cons of each device type and from that, perhaps, you'll be able to find the one that's right for you.

PPC Devices

Palm-size PCs are great for applications where

- Minimal data entry is required
- Heavy reading isn't required
- The lack of a PCMCIA slot is acceptable
- The information displayed can be made to fit in the 320 × 240 display area

If your future application is likely to be OK with those constraints, by all means, consider a PPC device. Their main advantage from the prospective CE developer's point of view, as we'll see in a moment, is their cost. As a development tool, the PPC devices are more demanding than an HPC/Pro because getting an application to look good on a PPC device requires more work than on either of the other platforms.

HPC Devices

HPCs offer a midrange option for applications where

- A keyboard is required
- A PCMCIA slot is required
- A larger screen is required

If that sounds like your future application, then an HPC is the way to go. One additional advantage of an HPC is that most HPCs are still running CE 2. If your program runs on an HPC running CE 2, it should run on all of the CE platforms.

HPC/Pro Devices

HPC/Pros occupy a unique niche in the array of portable devices available. They are near-laptops with just about everything a laptop offers. They're perfect for applications where

- A keyboard is required
- A PCMCIA slot is required
- A much larger screen is required

If your future application fits the bill, then you'll want to consider an HPC/Pro device. And if you don't already own a laptop, you'll want to *seriously* consider the HPC/Pro devices. As a laptop substitute, the HPC/Pro devices will likely prove very adequate for most uses. One thing to watch out for, though, is that HPC/Pros run CE 2.11, which offers a number of features not available on other versions of CE. In other words, if you want your application to compile for all of the 2.*x* versions of CE, you will have to be extra cautious when coding to make sure that you use only those functions available on all CE devices.

Cost

Table 13.6 shows the different approximate costs of each of the form factors.

TABLE 13.6: CE Form Factors and Their Approximate Costs

Form Factor	Cost
PPC	$200–$299
HPC	$400–$499
HPC/Pro	$899

As of this writing, most PPC devices are priced right around $300, but some have been priced as low as $200. This means that for a relatively small investment (the cost of a new hard drive, perhaps), you can have an actual CE device to test and develop with.

HPC devices are getting cheaper all the time, with the price hovering around the $400–$499 range, while HPC/Pros are still the most expensive of the lot, priced at around $899.

Conclusion of Study 3

The conclusion you make really depends on your own personal CE needs and plans. If you're just experimenting, the PPC devices are probably the best low-risk, low-investment strategy. If cost is not an issue, however, either the HPC or the HPC/Pro devices will serve you quite nicely.

Summary

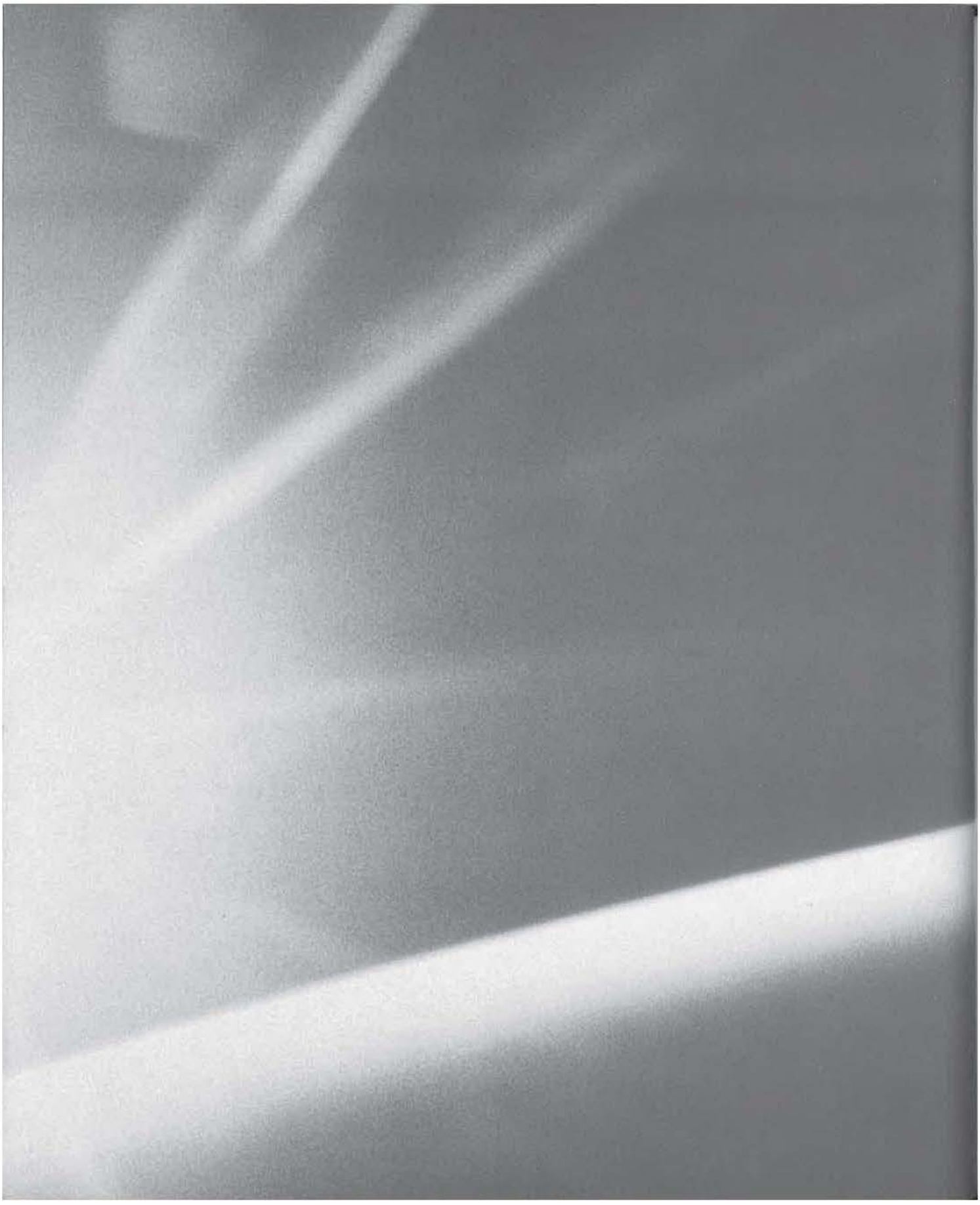
In this chapter, you saw at two examples of how CE devices can be used. In addition, you examined the pros and cons of each CE device so that you can pick the most appropriate CE device for your development purposes.

In the next chapter, you'll see how to put the finishing touches on a Windows CE application.

PART IV

Finishing Touches

- Chapter 14: Distributing Your CE Application
- Chapter 15: Microsoft's Logo Requirements



CHAPTER

FOURTEEN

14

Distributing Your CE Application

- Creating Help for Windows CE
- The Two Types of CE Help Files
- Installing Your Software

In this chapter, we'll explore what it takes to put the finishing touches on your CE applications. Specifically, you'll learn how to create help files for Windows CE, what the different types of help files are, and what the advantages and disadvantages of each one are. Then, to round out the chapter, we'll look at the available options as far as setup or install programs are concerned.

Creating Help for Windows CE

If you've ever written help files for Windows 3.*x*, 98, or NT, you know what a pain they can be... Some of their more notable problems are that

- The text portion of the help files has to be written in an arcane set of formatting commands known as RTF
- The pictures and text all have to be compiled into an HLP file
- If any portion of the help file is incorrect, the whole thing has to be recompiled from scratch

The whole process was much more complex and time-consuming than it needed to be. Thankfully, though, Windows CE has done away with the Windows 98/NT help-file system completely!

Instead, the CE help system uses a unique subset of HTML—the same set of universally supported formatting commands used to create Web pages.

TIP One of the nice things about an HTML-based help system is that testing and debugging your help is a much quicker process. That's because you can actually edit the help files *on the device* if you find a small mistake and want to correct it immediately. This means you can test your fix right away, without having the delay of copying the help file from the Desktop to the device.

The Two Types of CE Help Files

Just like the Windows 98/NT help system, which has the .cnt, or Contents, file and the .hlp file formats, Windows CE also has two help file-types:

1. .htc, or Help Contents file (comparable to the .cnt file of 98/NT)
2. .htp, or Help Topic file (comparable to the .hlp file of 98/NT)

NOTE A file extension of .html will also work for help files; in fact, this is the required file extension for PPC help files.

The HTC/Contents format is used, as you've probably guessed, to store a table of contents, almost like a home page that contains links to one or more Help Topics. Each of these Help Topics is then contained in a separate HTP/Topic file.

The HTC/Contents file has only one purpose, but the HTP/Topic files have two possible purposes:

- Use as a file containing one or more single topics (subpages of an HTC file)
- Use as a standalone, self-contained help file, with a reduced table of contents and multiple smaller topics.

One way to organize your program's help system is to use one HTC/Contents file and multiple HTP/Topic files. This is the way you'd probably organize your help system if you had, say, a complex program with multiple features in which the user might need additional instruction. The tradeoffs of this multiple-file help system can best be viewed in this way:

Multiple-File Help System	(HTC File Paired with One or More HTP Files)
Advantages	With minor tweaking, can be created with an off-the-shelf HTML editor.
	Topic files may be stored anywhere on device.
Disadvantage	More difficult to maintain and test.
	All files must be located in the \Windows directory.

The other way to organize your program's help system is to put everything (including a brief contents page) into one single HTP/Topic file. This is almost certainly the way you'd organize your help if you were writing an application on the scale of complexity close to a notepad-like application. The trade-offs of the single-file system can best be viewed in this way:

Single-File Help System	(Stand-Alone HTP File)
Advantage	Easier to maintain and test a single help file.
Disadvantages	Can't be easily created with an off-the-shelf HTML editor. See the "FrontPage and HTP Files" sidebar below for more information.
	Help files can be anywhere on the device, but links must then contain full paths, even if the link points to the same file in which it appears.
	The only exceptions to this is when the file is in the `\Windows` directory, in which case your internal links do not have to include the full path.

NOTE Most of the applications that are already on your CE device, including those written by Microsoft, use the single-file help system.

Let's examine each help system in detail, dissecting a sample system in each style.

The Single-File Help System

First, let's take a look at the simpler single-file help system, where all of the help topics and the contents are placed in a single HTP file.

At first glance, the source for one such HTP file might look a little like a Web site desperately in need of content. Figure 14.1 shows a simple HTP file, which has one Contents entry and one topic.

FIGURE 14.1:

A simple HTP help file, as it would be viewed on a CE device

This is the source of that same HTP file:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" Content="template.htp#contents">
<title>Template Single-File Help System</title>
</HEAD>
<BODY BGCOLOR=#FFFFFF TEXT=#000000>
<!-- PegHelp -->
<p>
<a name="contents"></a><b>The Contents</b>
</p>
<p>
<a href="template.htp#topic_1">Topic #1</a>
</p>
<!--now include a link (image) to next topic-->
<div align=right>
<a href="template.htp#topic_1">
<img src="arrowr.2bp" border=0></a>
</div>
<!-- PegHelp -->
<a name="topic_1"></a>
<b>Topic #1</b>
<!--now include a link (image) to the "main" page or topic-->
<div align=right>
<a href="template.htp#contents">
<img src="arrowl.2bp" border=0>
</a>
```

```
<!-- PegHelp -->
</BODY>
</HTML>
```

As you can see, it's not very complex. However, as with most CE development tasks, everything has to be done just right. So, in order to get a better understanding of the various parts of this help file, let's take it one step at a time.

The Header

Let's take a look at a typical CE help-file header, which looks very similar to the header of a standard HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" Content="template.htp#contents">
<title>Template Single-File Help System</title>
</HEAD>
<BODY BGCOLOR=#FFFFFF TEXT=#000000>
```

The main difference between an HTML document and an HTP document is that the HTP single-file help system requires that the `META HTTP-EQUIV "Content-Type"` tag contain the name of the help file (`template.htp`), followed by the pound sign (#), followed by the name or anchor of the first Contents entry. Hence the line:

```
<META HTTP-EQUIV="Content-Type" Content="template.htp#contents">
```

FrontPage and HTP Files

Incidentally, it is this customization of the `"Content-Type"` entry that makes it difficult to create single-file HTP help systems with off-the-shelf HTML editors such as Microsoft FrontPage 98. Although FrontPage will allow you to add, modify, and delete most other `HTTP-EQUIV` tags, it does not allow you to change the `"Content-Type"` entry. Like most documents, help files often require multiple revisions. The consequence of this is that each time you save your help file for testing, you must manually edit this line. As you can imagine, this can be quite annoying. That's why it's easier to simply use Notepad to create and edit the single-file system HTP help files.

The <!--PegHelp--> Comment

The next thing you'll probably notice about the HTP source is the line that reads:

```
<!-- PegHelp -->
```

Yes, this is just a fairly standard-looking comment tag, but the `PegHelp` text means that it serves a special purpose for the CE help viewer. The `<!-- PegHelp -->` comment is used as a separator between sections. It must also appear before the contents and after the final topic.

The next chunk of the HTP file is the only entry on the Contents page:

```
<p>
<a name="contents"></a><b>The Contents</b>
```

Here's where you could list the entire contents of your help file, along with some introductory text, perhaps:

```
</p>
<p>
<a href="template.htp#topic_1">Topic #1</a>
</p>
```

For the most part, this is fairly standard HTML. The `href` tag is what's of real interest here. It looks and works just like a standard `href` tag, and in the case of single-file HTP files, it's how you provide a link to a topic—`subtopic_1` in this case.

Next, you have a link to `subtopic_1`, this time via an image.

```
<!--now include a link (image) to next topic-->
<div align=right>
<a href="template.htp#topic_1">
<img src="arrowr.2bp" border=0></a>
</div>
```

Here again, this is fairly standard HTML, and all that's really worth noting is that the image is a 2-bit-per-pixel bitmap file, instead of a more traditional GIF or JPEG file format.

WARNING The Windows CE Help Viewer does not support GIF and JPEG images; only Windows CE bitmap types can be displayed in help files. Be especially careful of this if you're porting some existing HTML-based help documents to CE.

The Rest of the HTP file

The next section of the HTP file is the first actual topic of the help file, again, beginning with the <!-- PegHelp --> comment:

```
<!-- PegHelp -->
<a name="topic_1"></a>
<b>Topic #1</b>
```

Here's where you could provide some information about topic #1 to the user. For instance, maybe they need advice about how to use the help system.

Next, just as with the end of the Contents entry, you have a link (again via an image), this time taking the user back to the previous topic:

```
<!--now include a link (image) to the "main" page or topic-->
<div align=right>
<a href="template.htp#contents">
<img src="arrowl.2bp" border=0>
</a>
```

Finally, you have a closing <!-- PegHelp --> comment, followed by the standard HTML closing tags.

```
<!-- PegHelp -->
</BODY>
</HTML>
```

Additional HTP File Tricks!

There's one additional trick that you can make your HTP files do, and that is to display some static text (or links) at the beginning of every topic in the help file.

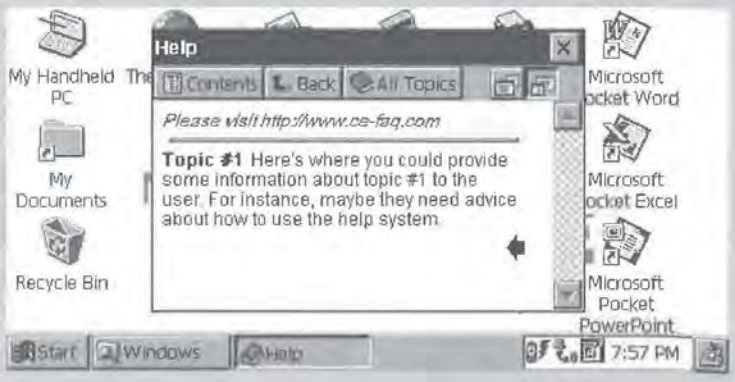
Simply by inserting any valid HTP/HTML tag after the <BODY BGCOLOR=#FFFFFF TEXT=#000000> tag and before the first <!-- PegHelp --> comment, you can cause that text to appear as the first thing on every page in the help file.

You might want to exploit this trick to let the user know that they can always find the latest information on your Web site, as in this example:

```
<!-- ... -->
<BODY BGCOLOR=#FFFFFF TEXT=#000000>
<i>Please visit http://www.ce-faq.com</i><hr>
<!-- PegHelp -->
<!-- ... -->
```

Continued on next page

When viewed on a CE device, the HTP file looks like this:



WARNING Don't forget: The single-file help system requires you to either use full paths in your links or that you put all of your HTP files in the `\Windows\` directory!

The Multiple-File Help System: The HTC File

The layout of the multiple-file help system is even closer to standard HTML than the HTP file you just examined. And, the best part about the multiple-file help system is that the files are so close to standard HTML that you can create and edit them using an off-the-shelf HTML editor such as FrontPage 98.

To develop a multiple-file help system, you must first create an HTC/Contents file. Figure 14.2 shows what a fairly standard HTC file might look like when viewed on a CE device.

Here's what the source code for that HTC file looks like:

```
<html>
<head>
<title>This is the Main Topic</title>
<meta http-equiv=refer content="">
</head>
<body>
<h3>This is the Table of Contents for our help file!</h3>
```

```
<p><a href="file://this_is_topic1.htp">this way to topic #1</a></p>
<p><a href="file://this_is_topic2.htp">or topic #2, if you
prefer</a></p>
</body>
</html>
```

FIGURE 14.2:

A standard HTC as viewed on a CE device

Probably the first thing you noticed about this HTC file is that it's even simpler than the HTP file you just examined. But just to be sure that you understand everything in it, let's once again take it apart, one section at a time.

The Header

First, let's take a look at a standard HTC file header:

```
<html>
<head>
<title>This is the Main Topic</title>
<meta http-equiv=refer content="">
</head>
```

The only point of real interest in this header is the fact that it's truly a standard HTML header. Unlike the HTP file you just looked at, this one has nothing unique or custom about it, and that's one of the main reasons it's possible to create these files with a standard HTML editor.

The Table of Contents

Now you come to the Table of Contents section for your HTC file. This is literally the body of your document:

```
<body>
<h3>This is the Table of Contents for your help file!</h3>

<p><a href="file://this_is_topic1.htp">this way to topic #1</a></p>
<p><a href="file://this_is_topic2.htp">or topic #2, if you
prefer</a></p>
</body>
</html>
```

In this example, the Table of Contents consists of a line announcing the table of contents:

```
<h3>This is the Table of Contents for your help file!</h3>
```

This is then followed by links to the two topics in our minimal help system:

```
<p><a href="file://this_is_topic1.htp">this way to topic #1</a></p>
<p><a href="file://this_is_topic2.htp">or topic #2, if you prefer</a></p>
```

The important point about these links is that they're just standard HTML code. This is just another reason why the multiple-file help systems are much easier to create.

Also, note that because there really aren't any topics, there aren't any of the `<!-- PegHelp -->` tags which pepper the single-file help system's HTP files.

TIP Although Microsoft's sample help files show tags such as `<a href="file://this_is_topic1.htp">` omitting the double slashes after `"file:"`, this does not appear to matter.

Table of Contents Entry Tags

In this sample HTC file, the Table of Contents entries are listed as individual paragraphs, as demonstrated by the use of the and tags. However, which tags you use is largely a matter of personal preference, and you can use any number of tags to list the entries in your Table of Contents.

Continued on next page

Officially, Microsoft's documentation recommends the use of the <menu> tag, which can be used to list any number of items so that they are indented and appear as one block of text. The following illustration shows what your simple HTC file would look like if you had used the menu tag instead of individual paragraphs.

It's also worth noting that Microsoft and many others do not use the <menu> tag at all but instead choose *list tags*, such as <UL> and <LI>.

Now that we've seen how simple an HTC file can be, let's see how simple a multiple-file help system's HTP files can be.

The Multiple-File Help System: The HTP File

When creating a multiple-file help system, you can use HTP files to contain

- multiple topics per file, *or*
- one topic per file.

If you choose the first option, the document source of your HTP file will end up looking exactly like a single-file help system's HTP file—in fact, it will look exactly like the one featured in the second section of this chapter. This, of course, means that you're back to HTP files that can't easily be created with off-the-shelf HTML editors. In other words, you could end up losing all of the advantages of the multiple-file help system.

However, if you choose the second option, the document source of your HTP file will look very different from the single-file help system's doocument source. Figure 14.3 shows one such HTP file as it looks in the CE Help Viewer.

FIGURE 14.3:

Multiple-file help system HTP file

The source for this document is amazingly simple:

```
<html>
<head>
<meta http-equiv="refer" content="file://the_contents.htc">
<title>some stuff</title>
</head>
<body>

<b>topic #1</b><br>

<p>Here's some information about
http://www.ce-faq.com</p>
</body>
</html>
```

The first point you'll notice about this HTP file is that it is almost entirely standard HTML formatting commands. In fact, there's only one line in the entire file that's CE-specific: an entry in the `<meta>` tag section of the header.

The Custom *<Meta>* Tag

The FrontPage 98 help defines `<meta>` tag as, "An HTML tag that must appear in the `<head>` portion of the page. `<Meta>` tags supply information about a page but do not affect its appearance."

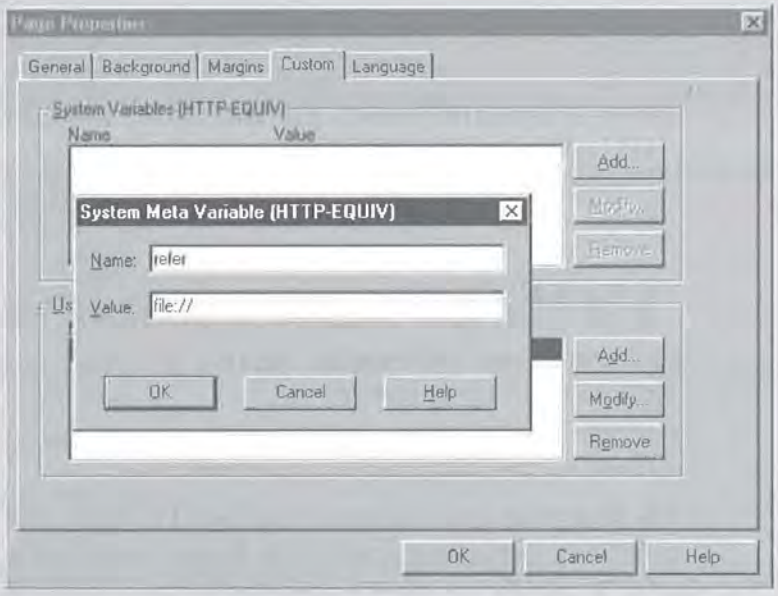
The <meta> tag in your HTP file looks like this:

```
<meta http-equiv="refer" content="file://the_contents.htc">
```

In this case, your <meta> tag is used to tell the CE Help Viewer that your HTP file refers to a certain HTC file; specifically, the_contents.htc. Aside from that, everything else in the HTP file is standard HTML and can easily be generated with FrontPage 98 or any other HTML editor.

Creating the Custom *<Meta>* Tag / *HTTP_EQUIV* Values within FrontPage 98

1. Open an existing HTP document, or create a new document.
2. Right-click on the document and select Page Properties from the pop-up menu.
3. Click the Custom tab, as shown below.



4. Click the Add button. The System Meta Value dialog box appears.
5. In the box marked Name, type **refer**.
6. In the box marked Value, type **file://**, followed by the CE path and file name of your contents file.
7. Click OK. That's it!

Now you know everything there is to know about CE help systems and how to create them. The next issue to tackle is the setup program—in short, how do you get your applications onto the end user's device?

Launching CE's Help Viewer from Your Application

Under Windows 98/NT, the help-viewer application is called `winhlp32.exe`; under Windows CE, the help-viewer application is called `peghelp.exe`. To launch the help viewer and cause it to bring up a specific help file, use code that looks something like this:

```
case WM_HELP: //handling a WM_HELP message
    PROCESS_INFORMATION pi;
    TCHAR szAppName[MAX_PATH] = TEXT("peghelp.exe");
    TCHAR szCmdLine[MAX_PATH] = TEXT("\\My App\\MyHelp.HTP");
    dwCreate = 0;
    CreateProcess(szAppName, szCmdLine, NULL, NULL, FALSE,
dwCreate, NULL, NULL, NULL, &pi);
```

Getting the Application to the User's Device

When it comes to installing your program on a user's CE device, you have two options.

The first option is to create a set of arcane, confusing, and generally difficult-to-use files and tools. If you decide to do this, you will need to create a setup program of your own design. This type of installation doesn't have to be fancy, though—it just needs to be able to copy files and launch another program. The advantage to this method is that it is free. The tools that you'll use, while difficult to master, are usually included with the CE toolkits or should be available from the Microsoft Web site. Let's call this option the Cab Wizard option, named for the tool you'll be using to create your installations, `cabwiz.exe`.

The second option is to use a graphical, familiar tool that allows you to create the setup in a few simple steps. You won't have to create your own setup program, and you'll have a complete, customized installation quickly and easily. This is the InstallShield for CE option.

In the following sections, we'll create a setup program for the TaskCE application developed in Chapter 5. First, we'll create a Cab file–based setup, and then we'll create an InstallShield-based setup.

The Cab Wizard Option

When it comes to the Cab Wizard option, there are three steps you need to complete in order to install your application on a user's CE device:

1. Create the Cab, or archive, files, one for each device or platform you've got an EXE for.
2. Next, create an INI file that contains some user-friendly information about your application, such as the application name, author, and so on.
3. Finally, create a Desktop PC-based setup program of your own.

Creating the Cab File

Cab files are a special kind of compressed archive file that Microsoft designed specifically for installations. To make Cab files for Windows CE installations, you'll have to use a program called Cab Wizard.

The creation of a Cab file can be broken down into two distinct steps:

- Creating an INF file to tell Cab Wizard which files to use
- Running Cab Wizard to create the Cab archive

Creating an INF File Cab Wizard is not your typical interactive wizard program; instead, it's a command-line utility that gets all of the information it needs from a complicated text file. It's this text file, known as the INF file, which you must create before you do anything else. The INF file is a basic settings file; a common INF file for creating CE applications has about a dozen sections, each one identified with a unique name enclosed in brackets, with several settings per section.

Before you start, note that this is the most complicated and confusing part of creating a Windows CE installation. While it's true that the INF file is basically an INI file, and therefore nothing you haven't seen before, it's also a good deal more complex than a standard INI file. For instance, there are several places in the INF file where settings in one section refer to settings in another section, which then refer to something else entirely. However, it should also be noted that CE developers create INF files regularly, so it isn't an impossible task.

The first section to be found in any INF file is the [Version] section, which usually looks like this:

```
[Version]
Signature = "$Windows NT$"
Provider = "doctorce.com"
CESignature = "$Windows CE$"
```

The Signature specifies the development machine; it is usually set to either "$Windows NT$" or "$Windows 95$". The Provider is the company or individual who wrote the software. Finally, the CESignature tells CabWizard that it must build CE-specific Cab files. This value is always set to "$Windows CE$".

The next section in an INF file is the [CEStrings] section. This is where to set up any paths or strings that you'll be using throughout the INF file. For example, your [CEStrings] section will look like this:

```
[CEStrings]
AppName = taskCE
InstallDir = %CE1%
```

TIP The reference to %CE1% is explained in detail in the "Predefined Destination Directories" sidebar below.

AppName and InstallDir are now available to be used throughout the INF file. This means that you can later specify the following elsewhere in the INF file:

```
SomeSetting=%InstallDir%\thefile.txt
```

AppName, of course, is the name of the application, and InstallDir is the destination directory on the device.

The Predefined Destination Directories

When you specified your InstallDir above, you used an identifier enclosed in percent symbols:

```
InstallDir = %CE1%
```

The %CE1% is part of a list of 17 predefined destination directories that you can use when creating your INF file. They can be used as the full path, as shown above, or they can be combined with string literals to specify a customized destination:

```
InstallDir = %CE1%\doctorce.com\TaskCE
```

Continued on next page

Obviously, the reason they're provided is to save you from retyping the more common paths and to prevent simple errors. These 17 predefined paths are listed in the following table:

Macro String	HPC Devices' Directory	Palm PC Devices' Directory
%CE1%	\Program Files	\Program Files
%CE2%	\Windows	\Windows
%CE3%	\Windows\Desktop	
%CE4%	\Windows\StartUp	\Windows\StartUp
%CE5%	\My Documents	\My Documents
%CE6%	\Program Files\Accessories	\Program Files\Accessories
%CE7%	\Program Files\Communication	\Program Files\ Communication
%CE8%	\Program Files\Games	\Program Files\Games
%CE9%	\Program Files\Pocket Outlook	
%CE10%	\Program Files\Office	
%CE11%	\Windows\Programs	\Windows\Start Menu\ Programs
%CE12%	\Windows\Programs\Accessories	\Windows\StartMenu\ Programs\Accessories
%CE13%	\Windows\Programs\ Communications	\Windows\StartMenu\ Programs\Communications
%CE14%	\Windows\Programs\Games	\Windows\StartMenu\Pro-grams\Games
%CE15%	\Windows\Fonts\Windows\Fonts	
%CE16%	\Windows\Recent	
%CE17%	\Windows\Favorites	\Windows\Start Menu

The next sections to create are the `[CEDevice.ChipType]` sections. These sections specify the chips you're targeting, so you need to create one `[CEDevice.ChipType]`

section for each chip. In addition, in each section you'll need to indicate the processor type(s) you're supporting:

```
[CEDevice.SH3]
ProcessorType = 10003
[CEDevice.MIPS]
ProcessorType = 4000
```

The next section we'll deal with is the `[DefaultInstall]` section. This is where to tell Cab Wizard which additional sections of the INF file are to be processed, regardless of what type of device the user has. Options that might appear here include registry settings, readme files, help files, and so on. Your `[DefaultInstall]` section for TaskCE merely contains a reference to a section that will contain information about a shortcut that you might like to create a little later on:

```
[DefaultInstall]
CEShortcuts = Shortcuts.All
```

It might also contain registry settings for both MIPS and SH3 installations.

The next sections you'll need to create are the `[DefaultInstall.ChipType]` sections. Just as you did with the generic `[DefaultInstall]` section, in the `[DefaultInstall.ChipType]` sections, you'll need to refer to later sections that contain settings for chip-specific files:

```
[DefaultInstall.SH3]
CopyFiles = Files.Common, Files.SH3

[DefaultInstall.MIPS]
CopyFiles = Files.Common, Files.MIPS
```

So, in the above example, `Files.Common`, `Files.SH3`, and `Files.MIPS` are all sections that appear later in the INF file.

Next, create a set of `[SourceDiskNames]` sections in which each source directory is identified with a unique number.

```
[SourceDisksNames]
1 = ,"Common Files",,'h:\"

[SourceDisksNames.SH3]
2 = ,"SH3 Files",,"h:\sh3"

[SourceDisksNames.MIPS]
2 = ,"MIPS Files",,"h:\mips"
```

Now that you've identified each source directory, you can tell Cab Wizard which files to add to the archive and where to find each file. You do this in three [SourceDiskFiles] sections:

```
[SourceDisksFiles]
 readme.txt = 1

[SourceDisksFiles.SH3]
taskCE.exe = 2

[SourceDisksFiles.MIPS]
taskCE.exe = 2
```

From the settings in these sections, Cab Wizard knows to look for the common file readme.txt in the directory marked 1 under the [SourceDiskNames] section above.

Next, tell Cab Wizard where you want all of these files to end up once they're on the CE device. This section, [DestinationDirs], is fairly self-explanatory. Each previously outlined section is assigned a destination directory where all of the files will eventually be copied:

WARNING The %InstallDir% path is subject to change; the only thing it really specifies is the default installation path, which the user may change when installing the software via App Manager.

```
[DestinationDirs]
Shortcuts.All = 0,%CE3%
Files.Common = 0,%InstallDir%
Files.SH3 = 0,%InstallDir%
Files.MIPS = 0,%InstallDir%
DefaultDestDir = 0,%InstallDir%
```

The Shortcuts.All entry specifies the destination for all shortcuts, the Files.Common specifies the destination for all common files, and so on.

NOTE The 0 that precedes each directory is a feature normally found in Desktop PC INF files, but Windows CE does not support this feature.

Next, you get to specify options for each of the files we're copying. In the [Files.Common] section, tell Cab Wizard to rename readme.txt to readthis.txt. The executables (in [Files.SH3] and [Files.MIPS]), however, will remain unchanged:

```
[Files.Common]
readme.txt,readthis.txt,,0

[Files.SH3]
taskCE.exe,,,0

[Files.MIPS]
taskCE.exe,,,0
```

Finally, tell Cab Wizard that you'll want a shortcut for your TaskCE application to be placed on the Desktop:

```
[Shortcuts.All]
taskCE,0,taskCE.exe
```

The entry under [Shortcuts.All] can be broken down in this way: the first item is the name of the shortcut, the 0 indicates that this is a shortcut to a file, and the last value specifies the name of the executable the shortcut is pointing to.

You can also specify a shortcut to a folder by changing the 0 to any other number. Similarly, you can specify that a shortcut be created in a directory other than that specified in [DestinationDirs] by simply appending your desired directory to the existing entry. So, in the above example, if you wanted that shortcut to appear in \Windows\Program Files\Accessories instead of the default directory, you would set it up as

```
[Shortcuts.All]
taskCE,0,taskCE.exe,%CE6%
```

That's the full picture on INF files. As you've probably realized, they're not very easy to create or friendly to use. It's generally regarded that once you create one that does what you need, you ought to simply use it as a template for all future installations. Naturally, the INF file we've created here is provided on the CD for this book so that you can use it as a template for your installations.

Running Cab Wizard to Create the Cab Archive Now that you've finished with your INF file, the next step is to run the Cab Wizard against your INF file so that it can create the Cab archives for you. Of the two steps, this is definitely the simpler one.

All you have to do to create your Cab files is open a command prompt window, go to the directory where `CabWiz.exe` is located and type:

```
CabWiz.exe "C:\TaskCE\TaskCE.INF" /err errors.txt /cpu mips sh3
```

The Cab files will be created for you and placed in the same directory as `CabWiz.exe`. In the case of your TaskCE project, the resulting Cab files will be named `TaskCE.SH3.cab` and `TaskCE.MIPS.cab`—one Cab file for each chip type specified after the `/cpu` command line option. If Cab Wizard cannot create the Cab files for any reason, the errors will appear in the `errors.txt` file you specified with the `/err` option.

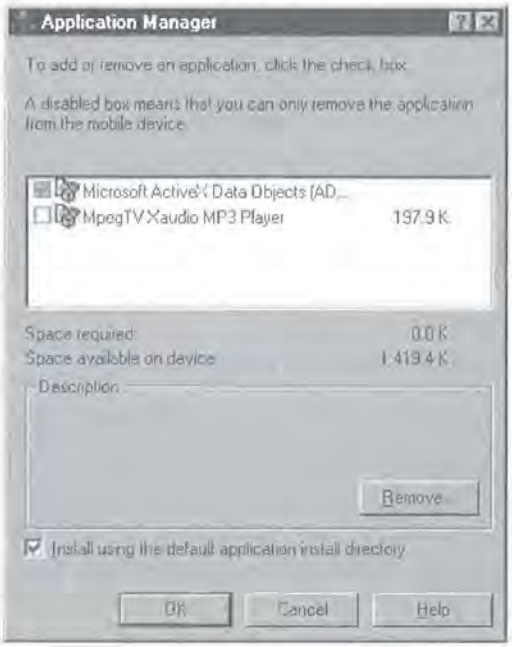
Now that your Cab file creation is completed, it's time to create an INI file.

Creating an INI File

The INI file you're going to create now is the INI file that will be used by your setup program to register your application with the CE Application Manager, shown in Figure 14.4.

FIGURE 14.4:

The CE Application Manager



This INI file has only two sections and can be created very quickly.

The first section, `[CEAppManager]`, specifies two values:

- The version of CE App Manager, which should always be 1.0
- The name of the next section in the INI file

Or, as it appears in the actual INI file:

```
[CEAppManager]
Version=1.0
Component=TaskCE
```

The second section, named for the `Component` setting of the first section (in your case `[TaskCE]`), specifies several additional values used by the CE Application Manager:

```
[TaskCE]
Description=TaskCE
CabFiles=taskce.sh3.CAB,taskce.mips.CAB
```

The `Description` setting specifies the text that CE Application Manager will display when the user highlights your application in the list of available programs. The `CabFiles` option tells CE Application Manager the names of your Cab files. The most important point to know about the `CabFiles` setting is that there should not be any spaces following the equal sign or the comma(s); a space on this line can actually cause your installation to fail!

As you can see, the INI file is very simple. Here again, it's a good idea to create one that works, and then use it as a template for future installations.

Creating Your Own Setup Program

The next step in the Cab file method of installing your software is to create your own setup program. The only real task that the setup program must perform is to copy the Cab files and the INI files to a subdirectory under the directory in which the CE Application Manager is located. We'll design your setup program to be as generic as possible so that when it's done, all you have to do is include it in the same directory as your INI and Cab files, and it will do the rest!

Although you can create your setup program using just about any tool, including batch files, let's use Delphi as a quick-and-dirty solution. We'll create a blank project without any forms so that what you're left with will be a single DPR file with a minimal `Uses` clause.

TIP Removing the forms and reducing the `Uses` clause keeps your EXE file size low.

The first thing your setup must do is locate your INI file, which should exist in the same directory as the setup program itself. Use the FindFirst function to search for the first INI file in your directory:

```
AppExePath := ExtractFilePath(ParamStr(0));
if (FindFirst(AppExePath + '*.ini', faAnyFile, SearchRec) <> 0) then
begin
```

If no INI file is found, report the error and quit:

```
  MessageBox(NULL, PChar('No Ini file found. Aborting.'),
PChar('Error'),MB_OK);
  Exit;
end
```

If the INI file was found, however, the next step is to locate the CE Application Manager. The way to do this is by querying the registry. When CE Services are installed on the user's machine, a set of values are added to the registry under `HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\Windows CE Services\`. One of these values in particular, `InstalledDir`, tells you where the CE Services files, including the CE Application Manager, are installed. Your next step, then, is to retrieve that value using Delphi's `TRegistry` object:

```
else
begin
  RegIni := TRegistry.Create;
  RegIni.CloseKey;
  RegIni.RootKey := HKEY_LOCAL_MACHINE;
  RegIni.OpenKey('\SOFTWARE\Microsoft\Windows CE Services\', False);
  //CEAppMgrPath := RegIni.ReadString('', '', '') + '\ceappmgr.exe';
  CEAppMgrPath := RegIni.ReadString('InstalledDir');
  RegIni.Free;
end;
```

The next step is to open the INI file you located earlier and retrieve the `Component` value from the `[CEAppManager]` section:

```
AppIni := TIniFile.Create(AppExePath + IniFileName);
DestinationDir := AppIni.ReadString('CEAppManager', 'Component', '');
AppIni.Free;
```

You'll use this value as the name of the subdirectory where you'll copy the INI and Cab files. If this value is empty or does not exist, report the error to the user and quit:

```
if DestinationDir = '' then
begin
  MessageBox(NULL, PChar('Error in INI file. Aborting.'),
PChar('Error'),MB_OK);
  Exit;
end;
```

Next, create the destination subdirectory (under the path of the CE Application Manager):

```
CreateDirectory(PChar(CEAppMgrPath + '\' + DestinationDir), nil);
```

Then loop through all of the files in the same directory as the setup program, copying them into the destination directory and making sure not to copy the setup program itself:

```
iResult := FindFirst(AppExePath + '*.*', faAnyFile, SearchRec);
while (iResult <> 0) do
begin
  if (SearchRec.Name <> ExtractFileName(ParamStr(0))) then
    CopyFile(PChar(AppExePath + SearchRec.Name),
      PChar(CEAppMgrPath + '\' + DestinationDir + '\' +
SearchRec.Name), TRUE);
  iResult := FindNext(SearchRec);
end;
FindClose(SearchRec);
```

Finally, launch the CE Application Manager, specifying the INI file you just copied as the command line parameter:

```
WinExec(PChar(CEAppMgrPath + '\ceappmgr.exe "' + CEAppMgrPath + '\' +
DestinationDir + '\' + IniFileName + '"'), SW_NORMAL);
```

TIP

If the CE Application Manager ever gives you an error message, you can turn on its debugging feature by adding the `/report` option to the command line just before the name of the INI file.

NOTE

It is also possible for a user to download a CAB file directly onto their CE device and install the software by double-tapping the actual CAB file. For more information, see Chapter 15.

And that's how to create a complete Cab file–based setup. While there are quite a few steps to manage and some of them are a bit confusing, many CE applications are installed in just this way without any additional tools.

However, if you prefer an easier route, there is a third-party tool that can be a big help.

InstallShield for Windows CE

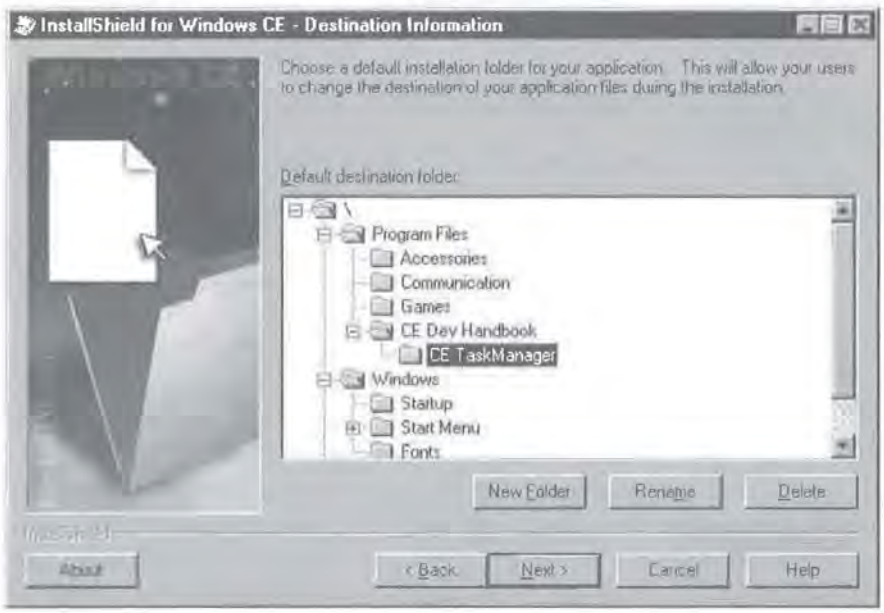
Install Shield for Windows CE is a tool that can help you to quickly and easily create installation files for Windows CE–based applications. InstallShield for Windows CE automatically knows about the different devices and chip types and makes it possible to target multiple platforms with ease.

When you start InstallShield and create a new project, you are first asked to enter some basic information about the application.

Next, the wizard asks you to specify the default destination folder of the application on the CE device. By default, InstallShield selects the `\Program Files\<Company Name>\<Application Name>` folder. You may specify another destination folder by selecting the desired destination in the TreeView, as shown in Figure 14.5.

FIGURE 14.5:

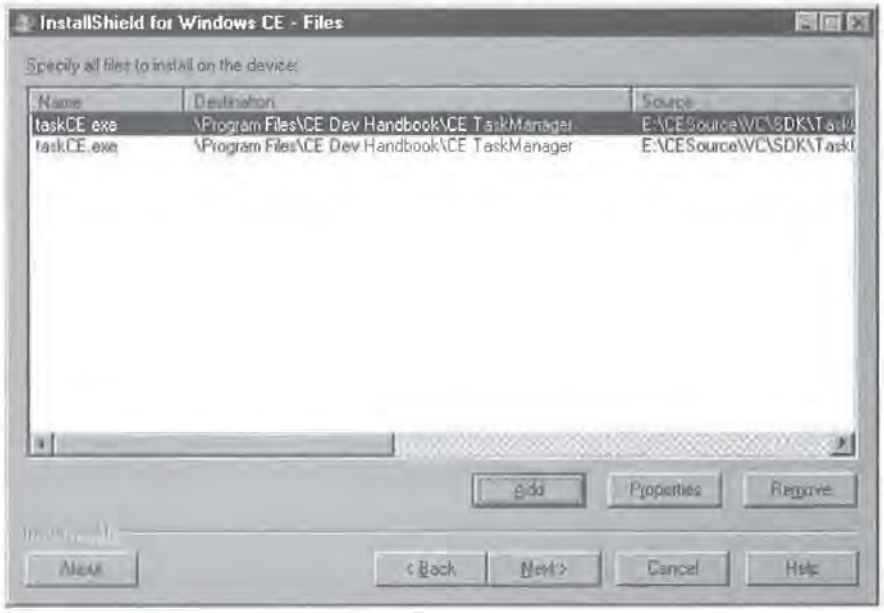
Specifying the default destination folder



Next, you're asked to specify all of the files you'll be installing to the user's device. The Files dialog, as shown in Figure 14.6, displays the name, destination folder, source folder, device type, supported processor type(s), and the size of each file you specify to include with your application. The Add File button allows you to add any files needed to run your application on the device.

FIGURE 14.6:

The Files dialog displays all of the files you'll be installing.



For your TaskCE application, browse to the location of the TaskCE files, select both the MIPS and SH3 files, and click Open. You can then add any additional files, such as the `readme.txt` file, in the same way.

In the next step, you can specify unique properties in the File Properties dialog for all of the files you've added using the Add File button. As shown in Figure 14.7, you can specify:

- Different destination folders
- Special copying options (i.e., Always update older file on target)
- Target chip type of file

In the next step, you can specify any shortcuts that you'll need. So, for your TaskCE project, you'll need to create a shortcut on the Desktop. Identify the shortcut target, the destination for the shortcut, and a name. You can rename shortcuts or change the destination folder of the shortcut by selecting the desired shortcut

from the list in the Shortcuts dialog and clicking the Properties button. The Shortcut Properties dialog is shown in Figure 14.8.

FIGURE 14.7:
The File Properties dialog

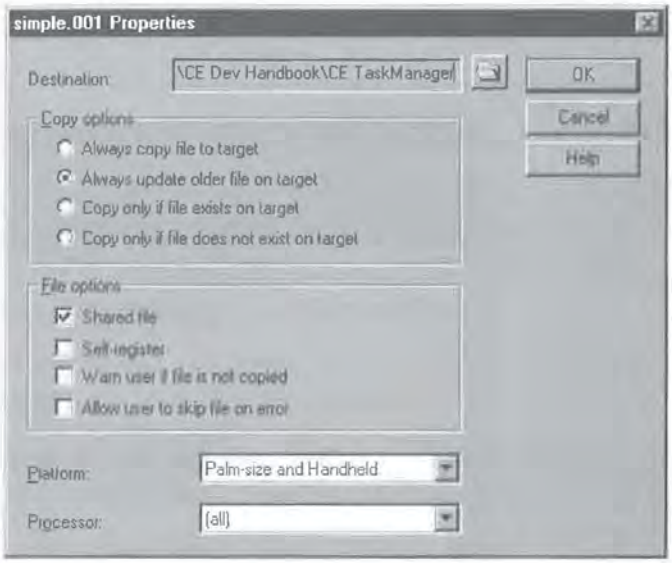
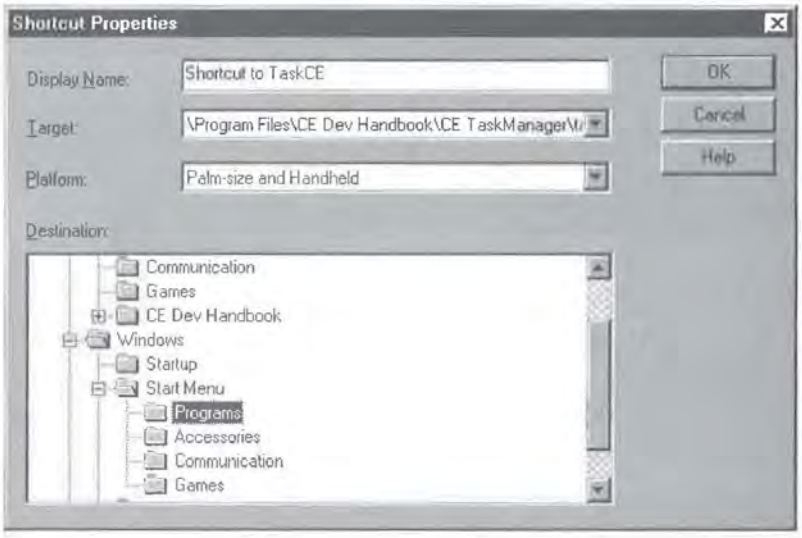


FIGURE 14.8:
The Shortcut Properties dialog

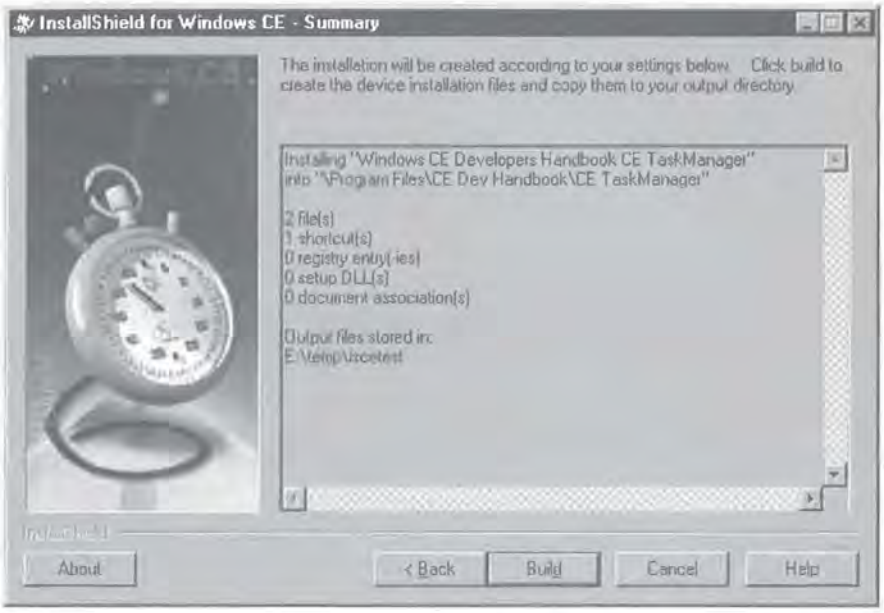


In the next few steps, InstallShield allows you to specify additional options for your application, such as file associations and registry keys. While TaskCE does

not have any file associations or registry keys associated with it, it's worth noting how easily these items can be configured with InstallShield.

Finally, when you're all finished setting up the application and any additional options, InstallShield displays a confirmation/summary of the options you've specified, as shown in Figure 14.9.

FIGURE 14.9:
The Summary or Confirmation dialog

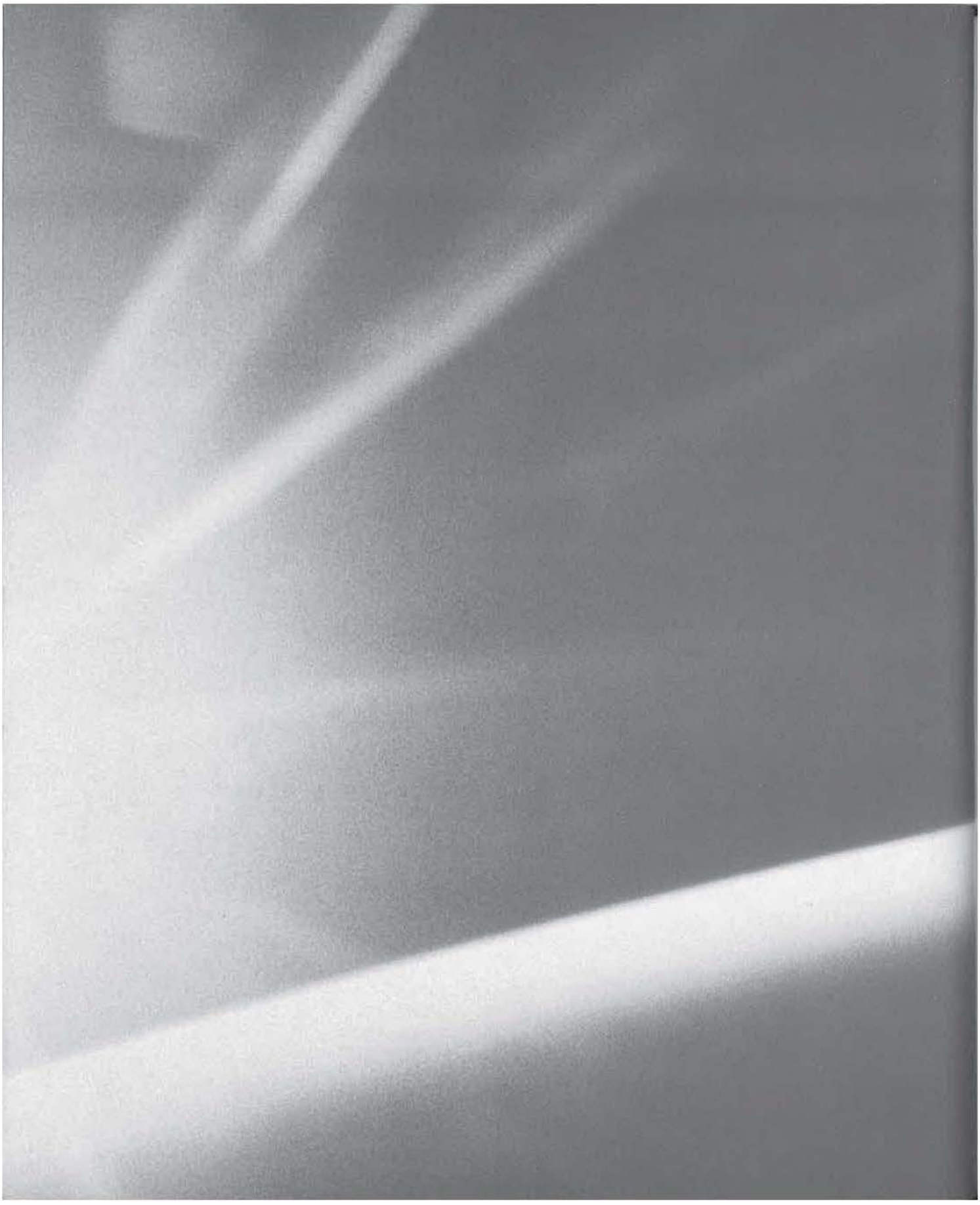


Confirm the options you've set, and InstallShield will create the INI, INF, and Cab files for you. You then have one set of distributable files that you can use for all of the chip types you're targeting.

Clearly, when it comes to CE installs, there's an easy way and a hard way to do it. You may prefer to create your own INF files so that you have more control over the install process, but the ease of using InstallShield for CE is a definite timesaver.

Summary

In this chapter, you learned how to put the finishing touches on a CE application. The help files are relatively easy to create, and there are a number of options that you can choose from to get the job done. CE installs, on the other hand, can be a bit time-consuming unless you use a third-party tool to help you.



CHAPTER

FIFTEEN

15

Microsoft's Logo Requirements

- Installation Requirements
- UI Requirements
- Functionality Requirements
- File-Handling Requirements

In this chapter, we'll be exploring the Logo process and what it takes to get your applications approved for the "Powered By Windows CE" Logo. The Logo program is often touted as a way for users to judge the quality of an application. In some sense, it may help to give the potential customer a sense that Microsoft or the powers that be have approved your application.

What Is the Logo Program and Why Should You Care?

The Logo program is not a computer program; rather, it is a type of program that developers can participate in. If your application conforms to these standards and rules, you can pay to have it certified by Microsoft's approved testing labs, Veritest. If your application passes the tests and is certified as conforming to the standards outlined by Microsoft, you earn the right to put the "Powered By Windows CE" logo on your application's box, advertisements, and so on.

TIP As of this writing, the cost to have an application tested by Veritest is $300 per application.

The general idea is that a potential customer for your software sees this Logo and concludes that your software has been tested and certified to be OK. As you'll see, the Logo is not a mark of quality or reliability of the software, but rather a certification that the program looks and feels a certain way.

NOTE *Logo'd* is the official Microsoft term for a product that has been verified as conforming to the Logo requirements.

Most of the freeware or shareware Windows CE programs do not conform to the Logo requirements. In fact, some of the Logo requirements may actually be in direct opposition to the wants of your users. For instance, it is required that certain Palm-size PC applications not show the file system or directory hierarchy in any way. For experienced users, not including an explorer or file manager-type application because of this requirement is extremely frustrating. To remedy this,

Casio provides a Find File utility, which can also be used to browse directories and files. Of course, this application would never be approved for a Logo, but it is extremely useful and probably quite popular.

Many other programs are already in compliance with the majority of the Logo requirements, and adding or modifying a few features here and there probably wouldn't be a major effort.

So What?

The real question is what benefit do you get from having your application Logo'd? It's certainly true that many potential customers may never notice the Logo on your program's box. Some may not care even if they do notice it. So why go to all of this trouble?

Well, of course, there is the argument that some potential customers do care. They want to know that launching your PPC application when there's already a copy running won't start two instances of it, but will instead bring the existing one to the foreground. They care that your HPC application supports mouse cursors, and that it will resize itself to account for the user's taskbar preferences.

However, the real benefits of the Logo program are in the perks that Microsoft provides to you. If you had your application certified for that reason alone, it would be worth the time and expense. Some of the more important benefits Microsoft provides for Logo'd applications include:

- Listing of your company and product on the Third-Party Showcase on the Microsoft Web site. This showcase is only for third parties that have qualified for the Logo.
- Microsoft quote provided if you are announcing in a press release that your company recently received the Logo.
- Discounts and special premium invitations to participate in the Windows CE-based Application CD samplers that are distributed to qualified Microsoft customers and at various trade shows.
- Invitation to participate in co-op advertising campaigns focused around Microsoft product launches.
- Invitation to participate in direct mail campaigns, joint seminars, and training events focused around Microsoft product launches.

- Opportunity to participate in Microsoft Partner Pavilions at trade shows, such as Windows World and COMDEX.
- Inclusion of your company and Logo'd product in the Windows CE-based Product Spotlight.
- Opportunity to participate in magazine advertisement discount offers of 10 percent (for new ad contracts only). To receive this discount, your product must be Logo'd and the Powered by Windows CE Logo must appear in the advertisement.

Clearly, there's a real, tangible benefit here that goes beyond just putting some additional art on a product's box. Assuming you've decided the Logo program is something you want to participate in, you then need to know what sort of modifications you'll have to make in order to get your programs to conform to the standards and rules.

In most cases, you won't have to do much of anything at all. If you've followed the UI guidelines and the model provided by the Microsoft applications, you'll probably be just fine. To see how the Logo rules will likely affect you, let's take a look at the most important ones for each platform, starting with the Palm-sized PC and then the Handheld PC (which includes HPC/Pro).

The Logo Requirements

For the rest of this chapter, we'll examine the four major areas in which the Logo requirements are likely to affect your application. They are:

- Installation
- UI Design
- Functionality
- File Handling

Although there are other sections to the Logo specifications, these are the items that are likely to affect you more than the others.

Installation

When it comes to installing your software on the user's device, there are a number of rules you need to follow to ensure that you get Logo'd. Most of them make good sense anyway, and the others are relatively minor.

1. Your application must install using CAB files and/or the CE Application Manager.

HPC	PPC
Required	Required

Simply put, Microsoft wants you to use their system of CAB files and the CE Application Manager to install applications to the CE Device.

NOTE When the user downloads a CAB file from the Internet directly to their device, the installation is performed by a CE-based utility called WCELoad.

2. Your application must register the CAB files for all processors with CE AppManager when installed from the desktop.

HPC	PPC
Required	Required

Along with Installation Rule #7, this rule's purpose is to make sure that CE AppManager knows about the CAB files for processors other than the one used in the device hooked up at the time of install.

3. All nonshared program files must be installed to your application's directory.

HPC	PPC
Required	Required

All EXE and DLL files must be installed to the application directory, as long as these files will be used by your program and your program alone.

4. All shared program files must be installed to the Windows directory.

HPC	PPC
Required	Required

All EXE and DLL files which might be used by more than one application should be installed to the `\Windows` directory.

NOTE In the official documents from Microsoft, Installation Rules #3 and #4 are combined into one rule for the PPC platform.

5. When your application is uninstalled, you must remove the user settings from the Windows CE registry

HPC	PPC
Required	Required

When your application is uninstalled, you must remove all traces of it from the registry.

6. When installing from the desktop to the device, you should first copy all of the files to a subdirectory of the CE AppManager Path.

HPC	PPC
Recommended	Recommended

This technique is demonstrated by the sample setup described in the previous chapter.

7. CAB files for all processors should be copied to the desktop when the user is installing from the desktop to the device.

HPC	PPC
Recommended	Recommended

The idea here is that even if the end user only has a MIPS device, you must copy the CAB files for the other processors to their desktop machine. Although

users typically only have one CE device, they could very easily upgrade or purchase a new device that uses a different chipset. Therefore, it's best to provide a way for them to install your software to their new device, if needed.

TIP MIPS R3910 binaries will run on all of the MIPS chips.

8. When allowing the user to install from the Internet to a device, you should instruct them on choosing the correct CAB file for their device.

HPC	PPC
Recommended	N/A

This rule applies only to HPC devices, as PPC devices are not configured to allow a direct Internet-to-device installation.

Determining the User's CPU Type over the Internet

Microsoft recommends that you set up your Web site to automatically choose the correct CAB file for the user. This is possible because Pocket IE sends a special HTTP variable indicating the CPU type of the device. To determine the User Agent CPU, you can reference the environment variable HTTP_UA_CPU.

For example, here is a simple Perl script that tells the user what type of browser and CPU their device is running:

```
#!/usr/local/bin/perl
print "Content-type: text/plain \n\n";
print "Your browser type is: ";
print $ENV{'HTTP_USER_AGENT'};
print "\n\nYour CPU type is: ";
print $ENV{'HTTP_UA_CPU'};
exit;
```

You could then easily use this HTTP_UA_CPU variable to direct the user's browser to the specific CAB file that's right for their device.

9. When your application is uninstalled, you must either remove the data files or alert the user to the fact that you did not remove these files.

HPC	PPC
Recommended	Recommended

If you choose not to delete these files, Microsoft recommends that you advise the user to move the data files to the desktop, then recopy them to the device as needed.

10. If you do remove user data files, you must alert the users and give them the option of saving these files.

HPC	PPC
Recommended	Recommended

As a corollary to Installation Rule #10, if you are going to delete the user's data files, you must give the users the option of copying these files to the desktop first.

11. You must use meaningful, unique names for program files.

HPC	PPC
Recommended	Recommended

The idea here is that you would like to avoid installing a shared DLL from your application to the Windows directory when a similarly named file from another vendor may already exist. So, naming your DLL "`OurCompany_OurApplication_OurDLL.DLL`" would be unique and, therefore, unlikely to conflict with any other vendor's DLLs.

12. You must support installation to a compact flash card.

HPC	PPC
N/A	Required

When your application is installed, the user will have the option (via CE App-Manager) of installing to a compact flash card. If the user chooses this option, you must copy as many files as possible to the compact flash card.

13. You must create a shortcut on the Start menu for your application.

HPC	PPC
Recommended	Required

When installing your application to a *PPC* device, you need to create a shortcut to your application under the `\Windows\Start Menu\Programs\` directory on the device. While it is a good idea to do this on both platforms, it is especially important on the PPC device, as there is no Explorer-type application by which the user could otherwise find and run your application once it is installed.

When installing to an *HPC* device, Microsoft recommends that you create shortcuts to your application in the `\Program Files\` directory instead of the desktop.

UI Requirements

In this section, we'll look at all of the UI and shell-related requirements. The rules in this section are designed to ensure that all Windows CE programs present a consistent look and feel.

1. The command bar and rebars must appear at the top of the screen.

HPC	PPC
Required	Required

Although Microsoft allows you to offer the option of hiding the command bar, they do require you to, by default, show the command bar and to place it at the top of your application's main window.

NOTE Games and other applications that wouldn't ordinarily have menus or toolbars may not be required to follow this rule.

2. Menus must appear in the left-hand portion of the command bar, while buttons must use the right-hand portion.

HPC	PPC
Required	Required

This organization can be seen on just about any of the Microsoft applications that ship with a CE device.

3. Your Help "?" button must be placed to the right of the screen, just to the left of the Close "X" button.

HPC	PPC
Required	N/A

This UI-related rule, as well as the next one, does not apply to PPC devices, as the help system on PPC devices works slightly different than on HPC devices. For the PPC version, see UI Rule #5.

4. Your application must allow the user to press Alt+H to bring up Help.

HPC	PPC
Required	N/A

This rule is intended to ensure that users will be able to click or tap the "?" button and receive help. If for some reason you do not want to offer this functionality, Microsoft gives you the option of adding a Help menu to your command bar, which could then be accessed via the Alt+H key combination.

Here again, this rule does not apply to PPC devices; for PPC devices, see UI Rule #5.

5. Your applications must not show a "?" Help button on the command bar.

HPC	PPC
N/A	Required

As you saw in the previous chapter, PPC users access the help system off of the Start menu. In order to be consistent, your applications should not offer any different means of accessing help. This, of course, is also why UI Rules #4 and #5 do not apply to applications running on PPC devices.

6. Standard menu items must appear in a set order.

HPC	PPC
Required	Required

UI Rules #6, #7, and #8 all go together. They are concerned with making sure that your applications present the same look and feel as other applications, at least as far as the command bar is concerned. This rule, for instance, says that if your application includes the standard File, Edit, etc. menus, you must arrange them in the usual set order, from left to right, of:

```
File, Edit, View, Insert, Format, Tools, Window
```

7. Standard buttons must appear in a set order.

HPC	PPC
Required	Required

This rule parallels UI Rule #6 and says that if your application includes the standard New, Open, etc. buttons, you must arrange them in the usual set order, from left to right, of:

```
New, Open, Save, Print
```

8. Standard functions must appear in a set order.

HPC	PPC
Required	Required

This rule also parallels UI Rule #6 and says that if your application includes the standard text-formatting functionality, the buttons or drop-down combos that provide that functionality must appear in the following order, from left to right:

```
Style, Font, Font Size, Bold, Italic, Underline
```

9. Users can not close the application themselves.

HPC	PPC
N/A	Required

This rule requires that your PPC applications not have a Close "X" button or a File ➢ Exit menu. The reason for this is that Microsoft does not want the user to have to worry about managing memory; instead, the operating system itself closes the applications as memory is needed.

TIP

Because the Close "X" button is probably more comfortable to you as a developer, you may want to include the Close button until you're all finished testing the application, then remove it before shipping.

Finding the Previous Instance

One of the consequences of UI Rule #9 is that every time the user starts an instance of your application, you must make sure you check to see that no other instance is already running. To do this, you can use `FindWindow()` from within your `WinMain()` function. For example:

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPTSTR    lpCmdLine,
                   int       nCmdShow)
{
    HWND hWnd;
```

Assuming that `szAppName` is your application's window class name, and `szTitle` is the caption of your application's main window, call `FindWindow()`:

```
    hWnd = FindWindow(szAppName, szTitle);
```

If `FindWindow()` finds a window matching your criteria, it will return a handle to that window. So, if `hWnd` contains a valid window handle:

```
    if IsWindow(hWnd)
    {
```

set the focus to that window, bringing it to the front, and then exit from the second instance:

```
      SetForegroundWindow(hWnd);
      return 0;
    }
```

10. Command bar buttons must offer tool tips.

HPC	PPC
Required	Required

All of the buttons—if there are any—on your command bar must offer tool tip hints so that the user can hold their stylus over the button for a few seconds and receive a "yellow balloon hint" as to that button's purpose. Further, while both the PPC and HPC platforms require these tool tips, Microsoft has rules on how the hints are to be worded for each platform.

10a. Hints for HPC Applications must give the button's function and its keyboard shortcut. In other words, a hint for a button on an HPC application that printed the application's data might read "`Print (Ctrl+P)`."

10b. Hints for PPC Applications must give the button's function *but not its keyboard shortcut*. In other words, a hint for a button on a PPC application that printed the application's data might read "`Print`" (notice the omission of "`Ctrl+P`").

11. Applications must provide small (16 x 16) and large (32 x 32) icons for the main executable and all registered file types.

HPC	PPC
Required	Required

This is a common sense rule that's something you'd want to do with any Windows application.

12. Your applications must not duplicate the functionality of the Address Book API.

HPC	PPC
Required	Required

This rule, which affects mostly PIM (Personal Information Manager) applications, is designed to conserve storage space on the CE Device. Simply put, Microsoft does

not want each PIM application to create its own database in addition to the Contacts application database that's already on the system. If all PIM applications use the same database, there will theoretically be less wasted storage space. Therefore, if your application performs any kind of contacts-related operations, you must use the existing Contacts database and the Contacts database functions.

13. Your applications must not duplicate the functionality of the msgstore API.

HPC	PPC
Required	Required

Just as UI Rule #12 requires you to use the Contacts database and related functions, this rule deals with messages and message-related applications. If you have an application that stores messages similar to e-mail messages, you must use the msgstore database and the related msgstore functions.

14. If your applications use the IR port, you must adhere to the IrDA standards.

HPC	PPC
Required	Required

The purpose of this rule is merely to ensure that all CE-based applications use the IrDA standards.

15. Your applications must use the hourglass or wait cursor whenever necessary.

HPC	PPC
Required	Required

Although the HP Jornada is the only CE device with a mouse, all CE devices support an hourglass or wait cursor, although it looks slightly different from a regular Windows 98/NT wait cursor. The CE wait cursor appears as a small window, about 32 × 32, which displays an animation of a wait cursor.

16. Your applications must resize themselves depending on the taskbar settings.

HPC	PPC
Required	Required

Because the user can always choose to auto-hide the taskbar, your application must be able to accommodate the taskbar settings and use the full screen area.

Working with the Taskbar

There are really only two things you must do to work with the taskbar and use the full screen area for our main window.

1. The first time you need to be concerned about the taskbar is when you create our main window.
2. The second time you need to be concerned about the taskbar is if the user changes their taskbar preferences while your application is running.

1. Creating the Main Window

When you create your application's main window, you should specify its dimensions as `0, 0, CW_DEFAULT, CW_DEFAULT`. Your call to `CreateWindow()`, then, might look like this:

```
hWnd = CreateWindow(szWindowClass, szTitle, WS_VISIBLE | WS_SYS-
MENU | WS_CAPTION, 0, 0, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL,
hInstance, NULL);
```

The `CW_DEFAULT` value on Windows CE ensures that your application will occupy the full screen area, taking into account the taskbar settings.

2. Watching for Taskbar Changes While Your Application Is Running

Since the user can also change their taskbar settings while your application is running, you need to be able to resize your window at runtime. The way to accomplish this is to handle the `WM_SETTINGCHANGE` message. When you receive this message, you can call `SystemParametersInfo()` and request the dimensions of the working area (i.e., the screen size

Continued on next page

minus the taskbar size). You can then call `MoveWindow()` to resize your window accordingly. Or, in code:

```
case WM_SETTINGCHANGE:
   RECT rcWorkArea;
   //Get the working area
   SystemParametersInfo(SPI_GETWORKAREA, 0, &rcWorkArea, 0);
   //resize the window
   return MoveWindow(hWnd, rcWorkArea.top, rcWorkArea.left,
rcWorkArea.right, rcWorkArea.bottom, TRUE);
   break;
```

With these two changes in place, your application will be able to perfectly adapt to the user's taskbar preferences, making your application that much closer to being Logo'd.

17. Your applications must support Alt+tap for pop-up menus and Alt+Enter for property sheets.

HPC	PPC
Required	N/A

While pop-up menus and property sheets are by no means required components of a Logo'd application, if the application does offer those items, Microsoft requires that the user be able to access them in a certain way.

NOTE This rule does not apply to PPC devices, as the default Input Panel keyboard has no Alt key.

TIP Chapter 3 has a demonstration of handing an Alt+tap event.

18. Your applications must support mouse cursors.

HPC	PPC
Required	N/A

On CE devices that have a mouse, your applications must accommodate the mouse, especially in regard to selection operations and right-clicking.

WM_RBUTTONUP and Pop-up Menus

As you saw in Chapter 3, the WM_RBUTTONUP message was removed from the CE API, simply because CE devices didn't have a mouse. However, with the advent of CE devices such as the HP Jornada, which does offer a mouse, the WM_RBUTTONUP message has been added back into the CE API. So, in order to get an application to compile for (and behave correctly on) multiple CE platforms with regards to right-click handling, you must put an #ifdef around your WM_RBUTTONUP message handler:

```
#ifdef WM_RBUTTONUP
    //if there's true mouse support...
    case WM_RBUTTONUP:
        PopupMenuHandler();
    break;
#endif
```

19. Your applications should use the default system font (9-point Tahoma) whenever possible.

HPC	PPC
Required	N/A

This is the simplest of the UI Rules, requiring you to do absolutely nothing when it comes to fonts. Tahoma (9-point) is the default system font, so as long as you don't specify any font settings, you'll be in compliance with this rule automatically.

20. Your dialogs should be centered on the screen.

HPC	PPC
Recommended	N/A

Microsoft recommends centering dialogs in the nontaskbar area of the screen to improve the overall look and feel of your applications.

21. Your dialogs shouldn't be any larger than 466 x 198.

HPC	PPC
Recommended	N/A

Since the smallest HPC resolution is 480 × 240, you can safely design your dialogs to be 466 × 198 and be assured that they'll look good on any of the HPC devices.

22. Your buttons should use a bold font style to improve readability.

HPC	PPC
Recommended	N/A

Microsoft recommends using a bold font on command buttons to improve readability and to be consistent with other HPC applications.

23. Your applications must deal with the showing/hiding of the Input Panel.

HPC	PPC
N/A	Required

Specifically, your windows and dialogs must be arranged so that an Input Panel does not hide any edit boxes. (Input Panels are assumed to be 80 pixels high for the purpose of this rule). It's worth noting that it is apparently OK to have an edit box that's hidden by an Input Panel, as long as the window has a scroll bar available so the user can scroll the edit box into view. This, in fact, is the way many Microsoft PPC applications work.

NOTE Chapter 1 demonstrates how to add and remove a scroll bar whenever the Input Panel's state changes.

24. If you choose not to provide a tray icon or Start menu shortcut to your applications, you must ensure that the applications use the desktop as their parent window.

HPC	PPC
N/A	Required

This rule seems to be somewhat pointless, considering that any application not providing a Start menu shortcut would be in direct conflict with Installation Rule #13.

25. Your applications should not display underlined accelerators (Alt+letter) or shortcut keys (Ctrl+letter) in menus or dialogs.

HPC	PPC
N/A	Recommended

It is perfectly acceptable to use underlined accelerators in HPC applications; this rule applies only to PPC applications where there is no Alt key on the standard Input Panel keyboard.

Similarly, although PPC keyboards do have the Ctrl key, it is usually a more complex process for the user to tap out Ctrl+X than it would be for them to go directly to that menu item. Therefore, it's recommended that you not display the shortcut keys in our menu items' text.

26. Toolbar buttons should have keyboard equivalents.

HPC	PPC
Recommended	Recommended

Despite UI Rule #10b, which says that you must not show keyboard shortcuts in your tool tips, Microsoft recommends that you provide keyboard equivalents just the same.

27. Your buttons should support pop-up help.

HPC	PPC
N/A	Recommended

Many of the PPC applications support pop-up help, so Microsoft recommends that, for consistency's sake, your applications should offer it, too.

28. Your applications' windows should not be overlapping or resizable.

HPC	PPC
Recommended	Recommended

Microsoft strongly recommends that applications take advantage of the full screen area to the maximum amount possible. Further, CE devices do not allow the user to resize windows. Therefore, it is recommended that none of your applications' windows overlap any existing windows.

29. Your applications should use HTML-based help.

HPC	PPC
Recommended	Recommended

As the Windows CE help viewer uses HTML-based help files, it's strongly recommended that your applications use HTML-based help, as opposed to using some other help system.

30. Your applications' windows should not display a title bar.

HPC	PPC
Recommended	Recommended

As screen real estate on CE devices is already somewhat limited, you should do everything you can to maximize the space you do have available. One of the modifications you can make is to eliminate title bars from your applications and simply avoid putting the name of the applications anywhere on the top of the main window.

31. Your applications' closing and launching should execute as quickly as possible.

HPC	PPC
Recommended	Recommended

The idea behind this rule is that your applications should mimic as closely as possible the "instant on" feature of a CE device. The user will come to expect everything on a CE device to be instantaneous and, to maintain this appearance, your applications should start and close immediately.

32. Your applications' ListView controls should allow for drag multi-select.

HPC	PPC
N/A	Recommended

Because it's somewhat difficult to press the Ctrl or Shift keys on a PPC device while selecting items in a ListView, Microsoft recommends that you allow the user to simply mark the items they want by pressing and dragging the stylus around the items they want to select.

NOTE This is the default behavior for a ListView control and will require no extra work on your part.

Functionality Requirements

Although the Powered by Windows CE Logo is not intended to be an indication that the application behaves as advertised, there are some functionality-related rules that an application must follow in order to be approved.

1. Your applications must run on all of the CE-supported CPUs.

HPC	PPC
Required	Required

Unless you've included some kind of chip-specific assembly language in your application, you should have no trouble at all meeting this requirement.

TIP The exception to this rule is that if there is no commercially available device that uses a certain CPU, then your application will not be tested and is not required to be compatible with that CPU.

2. Your applications must be fully functional when submitted for testing.

HPC	PPC
Required	Required

Microsoft requires that all of the parts of your application are working and functional at the time you submit the application for testing. This rule simply exists so that your application can be fully tested for compatibility with the rules.

3. Your applications' windows and graphical components must resize themselves based on resolution and screen size.

HPC	PPC
Required	N/A

For the most part, this rule is covered if you simply create your window using the default values for width and height, as illustrated in UI Rule #16. However, it may be necessary to incorporate, say, a scroll bar on the left of your applications' main forms, so that it will still be possible for the user to access all of the controls on a smaller screen.

TIP The reason this doesn't apply to PPC devices is that all of the PPC devices available have the same display size.

4. Your help files must be working when you submit our applications for testing.

HPC	PPC
Required	Required

Although it's not required that your help files actually be fully written, they must at least display the correct topic when the user asks for help.

5. Your applications must use RAPI where applicable.

HPC	PPC
Required	Required

Simply put, Logo'd applications may not use any custom means of communicating with desktop or companion applications.

6. Your applications must use Winsock where applicable.

HPC	PPC
Required	Required

Here again, where possible and applicable, your applications should use Winsock to communicate with desktop or companion applications.

7. Your application must respond to WM_HIBERNATE messages by reducing the amount of memory used.

HPC	PPC
Required	Required

The only element not specified by this rule is the *amount* by which your application should reduce its memory usage. However, Microsoft does recommend a 25 percent reduction.

8. Your applications must exit without user intervention.

HPC	PPC
N/A	Required

This special PPC-specific rule is related to UI Rule #9, regarding the fact that users should not be able to close the applications themselves. Because the PPC OS closes applications as memory is needed, it's important that your applications don't require any user intervention to be closed. In this way, the automatic closing of applications is seamless, and the user never knows what's going on behind the scenes.

9. Our applications must store and load their state.

HPC	PPC
N/A	Required

This rule is also related to the way in which the PPC OS closes applications automatically. Part of the way the automatic closing process is supposed to work is that when the user starts up your applications for the second time, everything must appear exactly as they left it—again, this is so that they don't know what's going on behind the scenes. It's up to you to ensure that your applications save and restore their state to carry off the illusion of everything appearing as they left it.

10. Your applications' dialogs must not obscure the Input Panel.

HPC	PPC
N/A	Required

This applies only to dialogs that are not full-screen dialogs. If the dialog is long enough that it will obscure some portion of the Input Panel regardless, it must be positioned so that it is flush with the top of the display area.

11. Your applications should capitalize the first letter of any text in an edit box.

HPC	PPC
N/A	Recommended

This rule is designed to make it easier for PPC users to enter data with the Input Panel, especially contact-type data, where the first character of the text is capitalized anyway.

12. Your applications should support color.

HPC	PPC
Recommended	Recommended

There is a wide variety of devices available today, and your applications should look good on all of them.

13. Your applications must support long file names in all file-related operations.

HPC	PPC
Required	Required

Where appropriate, your applications must support and use long file names for saving/opening files, displaying file names, and so on. Further, it is recommended that your applications hide the three-digit file extension.

Here are the tests that Microsoft recommends in order to verify that your applications properly handle LFNs. If your applications follow the rules of this table, then you're handling LFNs just fine.

What the user enters	What the result should be
`Test`	`Test.ext`
`Test test test test`	`test test test test.ext`
`Test 1234567890`[...to 250 chars]	`Test 1234567890[...].ext`
`test` (3 blank spaces @ beginning of file name)	`test.ext` (no blanks)
`test` (3 blank spaces @ end of file name)	`test.ext` (no blanks)
`test ; + , = [ ]`	`test ; + , = [ ].ext`

In addition, files with invalid names (i.e., those containing characters such as question marks) should not save.

File-Handling Requirements

This set of rules relates to applications that mainly open, create, and edit documents.

1. Your applications must use common dialog boxes whenever appropriate.

HPC	PPC
Required	Required

This is another common-sense rule that applies across all Windows platforms.

2. Your applications should display the name of the open document on their taskbar buttons.

HPC	PPC
Required	N/A

For an example of this, look at Pocket Word.

TIP This rule does not apply to PPC devices, as PPC applications do not have buttons in the taskbar.

3. Your applications should not be multi-instance.

HPC	PPC
N/A	Required

Again, in relation to UI Rule #9, your PPC-based applications must only allow a single instance of themselves to be running at once.

4. Your applications must not show the file system in any way.

HPC	PPC
N/A	Required

Perhaps in order to make PPC devices more accessible to some users, Microsoft is attempting to insulate them from having to know the file system of their device. At any rate, if you stick to the common dialogs for file access, you should be just fine with this rule.

5. Your applications should create a shortcut to a recently used document in the Start Menu ➢ Documents menu.

HPC	PPC
Recommended	N/A

Microsoft recommends that you create a shortcut for a recently used document under the `\Windows\Recent` directory so that it will appear on the Start Menu ➢ Documents menu.

6. Your applications should have their own MRU lists.

HPC	PPC
Recommended	N/A

In addition to adding recently opened files to the system's Documents menu, your application should also maintain its own MRU list.

7. Your applications should support IR transfer of files, if appropriate.

HPC	PPC
N/A	Recommended

As there is no Explorer-type application on PPC devices that offers this functionality at a system level, it's recommended that you provide this feature from within your applications.

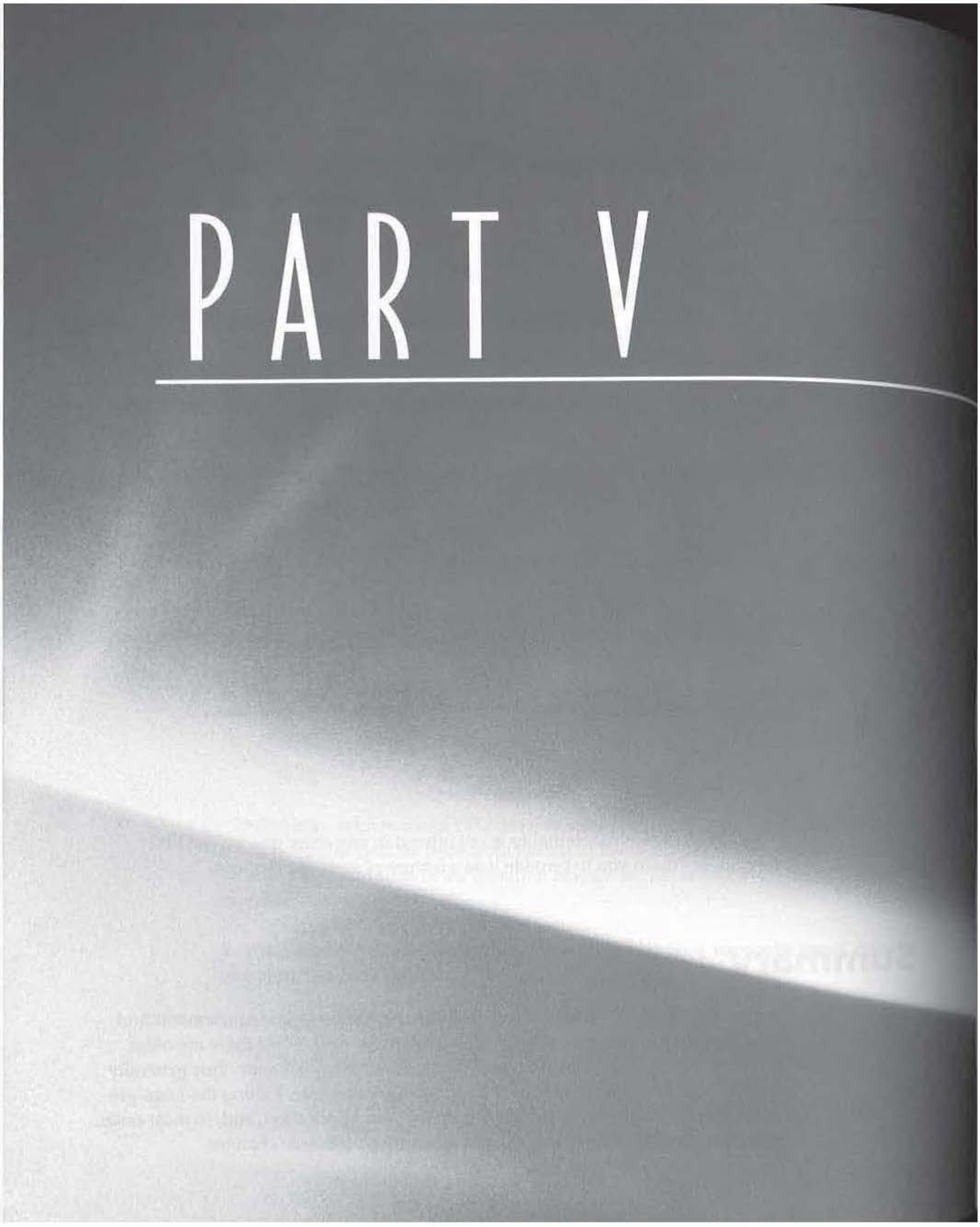
8. Your applications should offer the ability to e-mail files.

HPC	PPC
N/A	Recommended

Again, since this functionality is not offered in any other way on the PPC devices, it's up to you to provide it as a feature of your applications.

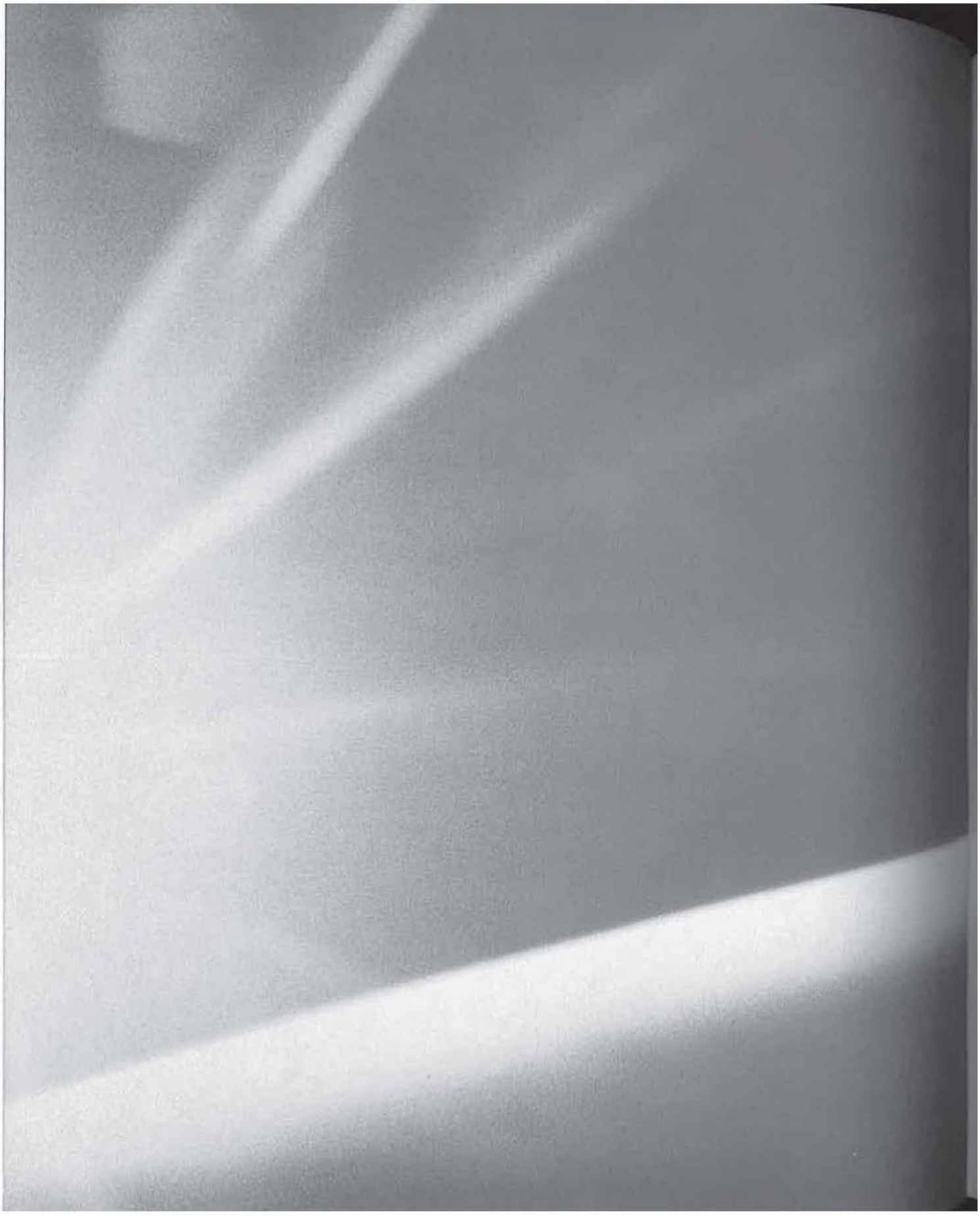
Summary

In this chapter, we've seen some of the more important Logo requirements and how they may or may not affect your applications. And, while there are other requirements, such as those for companion desktop applications, they generally won't affect the majority of the CE applications you create. Joining the Logo program is one of the best things you can do for your application, and, in most cases, shouldn't require more than $300 and some minor cosmetic changes.


PART V

Appendices

- Appendix A: The C Runtime Library Functions of Windows CE
- Appendix B: The CE 2.0 API



APPENDIX A

The C Runtime Library Functions of Windows CE

This appendix contains all of the C runtime library functions that are supported on all versions of Windows CE. In other words, you can safely use any of the functions listed here on any 2.*x* version of Windows CE. Although CE 2.11 supports many additional functions, they are of very little benefit to anyone who needs to target a wide range of CE devices. The format that we will use to describe these functions is as follows:

Function name

```
#include <Header file to be included>
```

Brief description

Example (if appropriate)

TIP

There are several places where Microsoft's online help contains an excellent example of some of these functions. In those cases, we have opted not to include an example, as it would only duplicate the help files. Instead, we refer you to the online help in a boxed tip like this one.

_cabs()

```
#include <math.h>
```

Returns a `double` containing the absolute value of a complex number.

Example:

```
struct _complex number = { 3.0, 4.0 };
double d;
d = _cabs( number );
printf( "The absolute value of %f + %fi is %f\n",
        number.x, number.y, d );
```

_chgsign()

```
#include <float.h>
```

Changes the sign of a `double` from positive to negative and vice versa.

Example:

```
_chgsign(-3.90923);  //returns 3.90923
```

_clearfp()

```
#include <float.h>
```

Clears the floating-point status WORD.

Example:

```
_clearfp();
```

_controlfp()

```
#include <float.h>
```

Sets the value of the floating-point status WORD.

Example:

```
_control87( _PC_64, MCW_PC ); //sets precision to 64 bit
```

_copysign ()

```
#include <float.h>
```

Returns the first value with the sign of the second.

Example:

```
_copysign(9.54, -8.6);  //returns -9.54
```

_ecvt()

```
#include <stdlib.h>
```

Converts a double numeric value to an ANSI string.

Example:

```
int dec, sign;
char *szNum;
szNum = _ecvt(3.7834, 4, &dec, &sign );  //szNum = "3.7834"
```

_fcvt()

```
#include <stdlib.h>
```

Converts a floating-point number to a string.

Example:

```
int dec, sign;
char *szNum;
szNum = _fcvt(3.7834, 4, &dec, &sign );  //szNum = "3.7834"
```

_finite()

```
#include <float.h>
```

Determines whether given double value is finite.

Example:

```
if (_finite(99.74))
{
    //...do something
}
else
{
    //...this code will never be reached
}
```

_fpclass()

```
#include <float.h>
```

Returns status word containing information on floating-point class.

Example:

```
double dVal;
// some code
int icls = _fpclass (dVal);
if (_FPCLASS_SNAN == icls || _FPCLASS_QNAN == icls)
{
   // show error, dVal is a Non-number
}
```

fpieee_flt()

Invokes user-defined trap handler for IEEE floating-point exceptions.

TIP The online help contains an excellent example of this function.

_fpreset()

Resets the floating-point package.

TIP The online help contains an excellent example of this function.

_gcvt()

```
#include <stdlib.h>
```

Given a floating-point value and the precision, converts the value to a string.

Example:

```
_gcvt(9.00213, 6, szDest);
```

_hypot()

```
#include <math.h>
```

Given floating-point lengths of two sides of a triangle, returns the hypotenuse.

Example:

```
fLengthC = _hypot(fLengthA, fLengthB );
```

_isnan()

```
#include <float.h>
```

Checks given double-precision floating-point value for not a number (NaN).

Example:

```
if (_isnan (fValue))
    {
        // handle non-number value, show error, etc..
    }
```

_itoa()

```
#include <stdlib.h>
```

Given an integer and the base (radix), converts the integer to an ANSI string.

Example:

```
_itoa(iNum, szBuff, 16);
```

_itow()

```
#include <stdlib.h>
```

Given an integer and the base (radix), converts the integer to a Unicode string.

Example:

```
_itow(iNum, szBuff, 16);
```

_j0()

```
#include <math.h>
```

Computes the Bessel function.

Example:

```
_j0(3.987);
```

_j1()

```
#include <math.h>
```

Computes the Bessel function.

Example:

```
_j1(3.987);
```

_jn()

```
#include <math.h>
```

Given the order of the function to compute, computes the Bessel function.

Example:

```
_jn(iOrder, 3.987);
```

_logb()

```
#include <float.h>
```

Extracts exponential value of `double` argument.

Example:

```
double dExp = _logb (7.775);
```

_lrotl()

```
#include <stdlib.h>
```

Left rotates a Long integer value by *n* bits.

Example:

```
_lrotl(0x759fe, nBitsToRotate);
```

_lrotr()

```
#include <stdlib.h>
```

Right rotates a Long integer value by *n* bits.

Example:

```
_lrotl(0x759fe, nBitsToRotate);
```

_ltoa()

```
#include <stdlib.h>
```

Given a Long integer and the base (radix), converts the integer to an ANSI string.

Example:

```
_ltoa(lVal, szNum, 16);
```

_ltow()

```
#include <stdlib.h>
```

Given a Long integer and the base (radix), converts the integer to a Unicode string.

Example:

```
_ltow(lVal, szNum, 16);
```

_matherr()

```
#include <math.h>
```

Handles math errors.

TIP The online help contains an excellent example of this function.

_memccpy()

```
#include <memory.h>
```

(or `string.h`)

Copies characters from a buffer until `iNumToCopy` characters have been copied, or until the `cToStopAt` character is found in the source.

Example:

```
char szWalrus[13] = "IAmTheWalrus\0";
char szCopy[13];
int iNumToCopy = 13;
char cToStopAt = "\0";
_memccpy(szCopy, szWalrus, cToStopAt, iNumToCopy
```

_memicmp()

```
#include <memory.h>
```

(or `string.h`)

Performs a case-insensitive comparison on `iNum` characters in two strings.

Example:

```
char szComp1[] = "Red house over";
char szComp2[] = "RED HOUSE OVER yonder";
int iNum = 14;
_memicmp(szComp1, szComp2, iNum);
```

_msize()

```
#include <malloc.h>
```

Returns size in bytes of a memory block allocated with `malloc()`.

Example:

```
iSize = _msize(pMemLocation);
```

_nextafter()

```
#include <float.h>
```

Returns next representable neighbor of x in the direction of y.

Example:

```
double x, y, result;
    x = 200.39434399999;
    y = 1000.99999;
    result = _nextafter(x, y);
```

WARNING This function seems to consistently return the value passed in as *x*. For instance, in this example, you would expect the function to return 200.394344; however, it seems to consistently return 200.39434399999.

_rotl()

```
#include <stdlib.h>
```

Left rotates an integer value by *n* bits.

Example:

```
_rotl (0x759fe, nBitsToRotate);
```

_rotr()

```
#include <stdlib.h>
```

Right rotates an integer value by *n* bits.

Example:

```
_rotr(0x759fe, nBitsToRotate);
```

_scalb()

```
#include <float.h>
```

Calculates the value of $x * 2^y$.

Example:

```
_scalb(x, y);
```

_statusfp()

```
#include <float.h>
```

Gets the floating-point status word.

TIP The online help contains an excellent example of this function.

_swab()

```
#include <stdlib.h>
```

Swaps *n* bytes from one byte or char array to another.

Example:

```
_swab(szSource, szDest, nBytesToCopy);
```

_ultoa()

```
#include <stdlib.h>
```

Given an unsigned Long integer and the base (radix), converts the integer to an ANSI string.

Example:

```
_ultoa(lVal, szNum, 16);
```

_ultow()

```
#include <stdlib.h>
```

Given a Long integer and the base (radix), converts the integer to a Unicode string.

Example:

```
_ultow(lVal, szNum, 16);
```

_wcsdup()

```
#include <string.h>
```

(or `wchar.h`)

Allocates a Unicode string buffer and then copies the contents of `szSource` into the new string. The new string will be created with `malloc()`, and it is the responsibility of the calling program to `free()` the string at some point.

Example:

```
szNewString = _wcsdup(szSource);
```

_wcsicmp()

```
#include <string.h>
```

(or wchar.h)

Compares lower-cased versions of two Unicode strings.

Example:

```
WCHAR szFirst[] = TEXT("Windows CE is great!");
WCHAR szSecond[] = TEXT('WINDOWS CE IS GREAT!");
iResult = _wcsicmp(szFirst, szSecond); //these strings will be
equal.
```

_wcslwr()

```
#include <string.h>
```

(or wchar.h)

Lowercases a Unicode string.

Example:

```
WCHAR szUpper[] = TEXT("WINDOWS CE IS GREAT!");
_wcslwr(szUpper);
```

_wcsnicmp()

```
#include <string.h>
```

(or wchar.h)

Performs a case-insensitive comparison on `iNum` characters in two strings.

Example:

```
WCHAR szComp1[] = "Red house over";
WCHAR szComp2[] = "RED HOUSE OVER yonder";
int iNum = 14;
_wcsicmp(szComp1, szComp2, iNum);
```

_wcsnset()

```
#include <string.h>
```

(or wchar.h)

Replaces `iNum` characters of a Unicode string with the character specified; see also `memset()`.

Example:

```
WCHAR szString[] = TEXT("Red house over");
_wcsnset(szString, TEXT('@'), 10);
```

_wcsrev()

```
#include <string.h>
```

(or `wchar.h`)

Reverses characters in a Unicode string.

Example:

```
WCHAR szString[] = "Red house over";
_wcsrev(szString);
```

_wcsset()

```
#include <string.h>
```

(or `wchar.h`)

Replaces all characters of a Unicode string with the specified character.

Example:

```
WCHAR szString[] = "Red house over";
_wcsset(szString, TEXT('@'));
```

_wcsupr()

```
#include <string.h>
```

(or `wchar.h`)

Uppercases a Unicode string.

Example:

```
WCHAR szUpper[] = TEXT("windows ce is great!");
_wcslwr(szUpper);
```

_wtoi()

```
#include <stdlib.h>
```

(or wchar.h)

Converts a Unicode string to an integer.

Example:

```
WCHAR szNum [] = TEXT("12345");
iNum = _wtoi(szNum);
```

_wtol()

```
#include <stdlib.h>
```

(or wchar.h)

Converts a Unicode string to a Long integer.

Example:

```
WCHAR szNum [] = TEXT("12345");
iNum = _wtol(szNum);
```

_y0()

```
#include <math.h>
```

Computes the Bessel function.

Example:

```
_y0(3.987);
```

_y1()

```
#include <math.h>
```

Computes the Bessel function.

Example:

```
_y1(3.987);
```

_yn()

```
#include <math.h>
```

Computes the Bessel function of order *x*.

Example:

```
_y0(x, 3.987);
```

abs()

```
#include <stdlib.h>
```

(or `math.h`)

Calculates the absolute value of an integer.

Example:

```
iAbsoluteVal = abs(-8000);
```

acos()

```
#include <math.h>
```

Calculates the arccosine of an angle of *x* radians.

Example:

```
fArcCos = acos(x);
```

asin()

```
#include <math.h>
```

Calculates the arcsine of *x* radians.

Example:

```
fArcSin = asin(x);
```

atan()

```
#include <math.h>
```

Calculates the arctangent of *x* radians.

Example:

```
fArcTan = atan(x);
```

atan2()

```
#include <math.h>
```

Calculates the arctangent of x/y radians.

Example:

```
fArcSin = atan2(y/x);
```

atoi()

```
#include <stdlib.h>
```

Converts an ANSI string to an integer.

Example:

```
char szNum [] = "12345";
iNum = atoi(szNum);
```

atol()

```
#include <stdlib.h>
```

Converts an ANSI string to a Long integer.

Example:

```
char szNum [] = "12345";
iNum = atol(szNum);
```

ceil()

```
#include <math.h>
```

Calculates the ceiling of a floating-point value.

Example:

```
fCeiling = ceil(x);
```

cos()

```
#include <math.h>
```

Calculates the cosine of x radians.

Example:

```
fCosine = cos(x);
```

cosh()

```
#include <math.h>
```

Calculates the hyperbolic cosine of *x* radians.

Example:

```
fHypCosine = cosh(x);
```

difftime()

```
#include <time.h>
```

Returns the difference between two `time_t` times.

Example:

```
time_t starting, ending, difference;
time(&starting);
//...do something time consuming
time(&ending);
difference = difftime(starting, ending);
```

div()

```
#include <stdlib.h>
```

Given numerator and denominator, calculates quotient and remainder, returning result in `div_t` structure.

Example:

```
div_t result;
result = div(3, 5); //3 divided by 5
```

exp()

```
#include <math.h>
```

Given a floating-point value, calculates the exponential.

```
y = exp(9.012847);
```

fabs()

#include <math.h>

Calculates the absolute value of an integer.

Example:

```
fAbsoluteVal = fabs(-4.36742);
```

floor()

#include <math.h>

Calculates the floor of a floating-point value.

Example:

```
fFloor = floor(x);
```

fmod()

#include <math.h>

Returns the modulus of a floating-point division.

Example:

```
fModulus = fmod(3.00, 5.00); //3.00 divided by 5.00
```

free()

#include <stdlib.h>

(or malloc.h)

Deallocates or releases memory allocated with malloc().

Example:

```
free(szDynamicString);
```

frexp()

#include <math.h>

Returns the mantissa and the exponent for a given floating-point value; performs the reverse operation of ldexp().

Example:

```
fMantissa = frexp(fFloatVal, &iExponent);
```

iswalnum()

```
#include <ctype.h>
```

(or `wchar.h`)

Returns `true` if the Unicode character falls in one of these ranges: A–Z, a–z, or 0–9.

Example:

```
bResult = iswalnum(TEXT('s'));
```

iswalpha()

```
#include <ctype.h>
```

(or `wchar.h`)

Returns `true` if the Unicode character falls in either of these ranges: A–Z or a–z.

Example:

```
bResult = iswalpha(TEXT('s'));
```

iswascii()

```
#include <ctype.h>
```

(or `wchar.h`)

Returns `true` if the Unicode character is an ASCII character.

Example:

```
bResult = iswascii('s');
```

iswcntrl()

```
#include <ctype.h>
```

(or `wchar.h`)

Returns `true` if the Unicode character is a control character.

Example:

```
bResult = iswcntrl(TEXT('\t'));
```

iswctype()

```
#include <ctype.h>
```

Tests the character c for the specified property.

Example:

```
int i = iswctype('D', _ALPHA|_UPPER);
```

iswgraph()

```
#include <ctype.h>
```

(or `wchar.h`)

Returns `true` if the Unicode character is a printable character and not a space.

Example:

```
bResult = iswgraph(TEXT('~'));
```

iswlower()

```
#include <ctype.h>
```

(or `wchar.h`)

Returns `true` if the Unicode character is a lowercase character.

Example:

```
bResult = iswalnum(TEXT('t'));
```

iswprint()

```
#include <ctype.h>
```

(or `wchar.h`)

Returns `true` if the Unicode character is a printable character.

Example:

```
bResult = iswprint(TEXT('*'));
```

iswpunct()

```
#include <ctype.h>
```

(or wchar.h)

Returns true if the Unicode character is a punctuation mark.

Example:

```
bResult = iswpunct(TEXT('!'));
```

iswspace()

```
#include <ctype.h>
```

(or wchar.h)

Returns true if the Unicode character is a space.

Example:

```
bResult = iswpunct(TEXT(' '));
```

iswupper()

```
#include <ctype.h>
```

(or wchar.h)

Returns true if the Unicode character is an uppercase character.

Example:

```
bResult = iswupper(TEXT('E'));
```

iswxdigit()

```
#include <ctype.h>
```

(or wchar.h)

Returns true if the Unicode character is a hexadecimal digit; that is, a character that falls in one of these ranges: 0–9, A–F, or a–f.

Example:

```
bResult = iswxdigit(TEXT('d'));
```

labs()

```
#include <stdlib.h>
```

(or `math.h`)

Calculates the absolute value of a Long integer.

Example:

```
lAbsoluteVal = abs(-4,568,992);
```

ldexp()

```
#include <math.h>
```

Given the mantissa and the exponent, calculates a floating-point value; performs the reverse operation of `frexp()`.

Example:

```
fFloatVal = frexp(fMantissa, &iExponent);
```

ldiv()

```
#include <stdlib.h>
```

Given numerator and denominator, calculates quotient and remainder, returning result in ldiv_t structure; Long integer version of `div()`.

Example:

```
ldiv_t result;
result = div(39873, 574359); //39873 divided by 574359
```

log()

```
#include <math.h>
```

Calculates the logarithm of a given floating-point value.

Example:

```
y = log(x);
```

log10()

```
#include <math.h>
```

Calculates the natural (or base 10) logarithm of a given floating-point value.

Example:

```
y = log(x);
```

longjmp()

```
#include <setjmp.h>
```

Restores stack environment and execution locale.

TIP The online help contains an excellent example of this function.

malloc()

```
#include <stdlib.h>
```

(or `malloc.h`)

Allocates or reserves memory blocks.

Example:

```
szString = (TCHAR *) malloc(100 * sizeof(TCHAR));
```

mbstowcs()

```
#include <stdlib.h>
```

Converts a sequence of multibyte or ANSI characters to a corresponding sequence of wide or Unicode characters.

Example:

```
mbstowcs(szWideString, szANSIString, strlen(szANSIString));
```

memchr()

```
#include <memory.h>
```

(or `string.h`)

Searches `iCount` characters of the `szString` buffer for the character `cChar`; returns a pointer to the first occurrence of the `cChar`.

Example:

```
szMatch = memchr(szString, cChar, iCount);
```

memcmp()

```
#include <memory.h>
```

(or `string.h`)

Compares `iCount` characters of two buffers.

Example:

```
iResult = memcmp(szStr1, szStr2, iCount);
```

memcpy()

```
#include <memory.h>
```

(or `string.h`)

Copies `iCount` characters from `szSource` to `szDest` buffers.

Example:

```
memcpy(szDest, szSource, iCount);
```

memmove()

```
#include <string.h>
```

(or `memory.h`)

Copies `iCount` characters from `szSource` to `szDest` buffers.

Example:

```
memmove(szDest, szSource, iCount);
```

memset()

```
#include <string.h>
```

(or `memory.h`)

Replaces `iNum` characters of an ANSI string or char buffer with the character specified; see also `_wcsnset()`.

Example:

```
char szString[] = "Red house over";
memset(szString, '@', 10);
```

modf()

```
#include <math.h>
```

Extracts the integer portion and decimal portion of a floating-point number.

Example:

```
fDecimalPortion = modf(fFloatVal, &iIntPortion);
```

pow()

```
#include <math.h>
```

Returns *x* to the *y* power.

Example:

```
pow(x, y);
```

qsort()

```
#include <stdlib.h>
```

Performs a quick sort.

Example:

```
qsort (infoarray, numelements, sizeof(DWORD), fnCompare);
```

rand()

```
#include <stdlib.h>
```

Returns a pseudorandom number.

Example:

```
iWinningLottoNum = rand();
```

realloc()

```
#include <stdlib.h>
```

(or `malloc.h`)

Reallocates memory blocks originally allocated with `malloc()`.

Example:

```
realloc(szBuffer, iNewSize);
```

sin()

```
#include <math.h>
```

Calculates the sine of *x*.

Example:

```
sin(x);
```

sinh()

```
#include <math.h>
```

Calculates the hyperbolic sine of *x*.

Example:

```
sinh(x);
```

sqrt()

```
#include <math.h>
```

Calculates the square root of a given floating-point value.

Example:

```
sqrt(4.00);
```

srand()

```
#include <stdlib.h>
```

Seeds the random number generation function, `rand()`, with a starting value.

Example:

```
srand(8945373);
```

strcat()

```
#include <string.h>
```

Concatenates or adds the contents `szString2` to the end of `szString1`, where `szString1` and `szString2` are both ANSI strings.

Example:

```
strcat(szString1, szString2);
```

strchr()

```
#include <string.h>
```

Finds an ANSI character in an ANSI string and returns a pointer to the first occurrence of that character in the string.

Example:

```
strchr(szString, 'c');
```

strcmp()

```
#include <string.h>
```

Compares contents of two NULL-terminated ANSI strings; performs a case-sensitive comparison.

Example:

```
iResult = strcmp(szStr1, szStr2);
```

strcpy()

```
#include <string.h>
```

Copies the contents of one ANSI string to another.

Example:

```
strcpy(szDest, szSource);
```

strcspn()

```
#include <string.h>
```

Searches an ANSI string szToSearch for an ANSI substring szSub; returns an integer specifying the number of characters in szToSearch before one of the characters in szSub occurs.

Example:

```
char szToSearch[] = "789areyou?";
char szSub[] = "eyou?";
strcspn(szToSearch, szSub); //result will be 5
```

strlen()

```
#include <string.h>
```

Returns the length in bytes of an ANSI string.

Example:

```
strlen("Hello, World!"); //result will be 13
```

strncat()

```
#include <string.h>
```

Appends at most iCount characters from ANSI string szSource to ANSI string szDest.

Example:

```
strncat(szDest, szSource, iCount);
```

strncmp()

```
#include <string.h>
```

Compares iCount characters of two ANSI strings.

Example:

```
iResult = strncmp(szStr1, szStr2, iCount); //compare the first
 iCount characters
```

strncpy()

```
#include <string.h>
```

Copies `iCount` characters from ANSI string `szSource` to ANSI string `szDest`.

Example:

```
strncpy(szDest+5, szSource, iCount); //copy first iCount characters
of szSource to szDest, starting at szDest's 5th character
```

strstr()

```
#include <string.h>
```

Finds an ANSI substring `szSub` in an ANSI string `szString`; returns a pointer to the first occurrence of `szSub` in `szString`.

Example:

```
szFirstOccurrence = strstr(szString, szSub);
```

strtok()

```
#include <string.h>
```

Parses an ANSI string by searching for single ANSI character delimiters in the string; returns a pointer to the next token.

Example:

```
char szStrToParse[] = "Windows CE version 2.11";
char* szToken = strtok(szStrToParse, " "); //szToken will point to
"Windows"
szToken = strtok(NULL, " "); //szToken will point to "CE"
//...etc.
```

swprintf()

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

(or `wchar.h`)

A `printf()`-like function that "prints" the data elements to a Unicode string instead of a console.

Example:

```
swprintf(szMsg, TEXT("%d:%d -%s"), 5024, 898, szTemp);
```

swscanf()

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

(or `wchar.h`)

Reads data elements from a formatted Unicode string; essentially performs the reverse function of `swprintf()`.

Example:

```
    swscanf("3 9 4 yes!", "%d %d %d %s", iInt1, iInt2, iInt3,
szString); //iInt1, etc. will all be filled with correct values from
the string
```

tan()

```
#include <math.h>
```

Calculates the tangent of *x*.

Example:

```
    fTangent = tan(x);
```

tanh()

```
#include <math.h>
```

Calculates the hyperbolic tangent of *x*.

Example:

```
    fHypTangent = tanh(x);
```

towlower()

```
#include <ctype.h>
```

(or `wchar.h`)

Converts a Unicode character to lowercase; characters already lowercased are ignored.

Example:

```
    towlower('P');
```

towupper()

```
#include <ctype.h>
```

(or wchar.h)

Converts a Unicode character to uppercase; characters already uppercased are ignored.

Example:

```
towupper('P');
```

vswprintf()

```
#include <stdlib.h>
```

Writes formatted output using a pointer to a list of arguments.

Example:

```
WCHAR wszBufferX[100];
mywsprintf (wszBufferX, TEXT("Number: %u, Text: %s"), 1234, 31415,
TEXT("Hello"));

    //wraps a wsprintf function
    void mywsprintf (LPWSTR pszTarget, LPWSTR pszFormat, int iCustom-
Value, ...)
    {
        va_list varlist;
        va_start (varlist, iCustomValue);
        vswprintf (pszTarget, pszFormat, varlist);
    }
```

wcscat()

```
#include <string.h>
```

(or wchar.h)

Concatenates or adds the contents szString2 to the end of szString1, where szString1 and szString2 are both Unicode strings.

Example:

```
wstrcat(szString1, szString2);
```

wcschr()

```
#include <string.h>
```

(or wchar.h)

Finds a Unicode character in a Unicode string and returns a pointer to the first occurrence of that character in the string.

Example:

```
wcschr(szString, TEXT("c"));
```

wcscmp()

```
#include <string.h>
```

(or wchar.h)

Compares contents of two Unicode strings; performs a case-sensitive comparison.

Example:

```
iResult = wcscmp(szStr1, szStr2);
```

wcscpy()

```
#include <string.h>
```

(or wchar.h)

Copies the contents of one Unicode string to another.

Example:

```
wcscpy(szDest, szSource);
```

wcscspn()

```
#include <string.h>
```

(or wchar.h)

Searches a Unicode string szToSearch for a Unicode substring szSub; returns an integer specifying the number of characters in szToSearch before one of the characters in szSub occurs.

Example:

```
WCHAR szToSearch[] = TEXT("789areyou?");
WCHAR szSub[] = TEXT("eyou?");
wcscspn(szToSearch, szSub); //result will be 5
```

wcslen()

```
#include <string.h>
```

(or wchar.h)

Returns the length in characters of a Unicode string.

WARNING You'll recall that Unicode characters are actually two bytes wide, which means that you must multiply the return value of `wsclen()` by 2 if you want to determine the number of bytes the Unicode string occupies.

Example:

```
wcslen(TEXT("Hello, World!")); //result will be 13
```

wcsncat()

```
#include <string.h>
```

Appends at most `iCount` characters from Unicode string `szSource` to Unicode string `szDest`.

Example:

```
wcsncat(szDest, szSource, iCount);
```

wcsncmp()

```
#include <string.h>
```

(or wchar.h)

Compares `iCount` characters of two Unicode strings.

Example:

```
iResult = wcsncmp(szStr1, szStr2, 7); //compare the first 7 characters
```

wcsncpy()

```
#include <string.h>
```

(or wchar.h)

Copies iCount characters from Unicode string szSource to Unicode string szDest.

Example:

```
wcsncpy(szDest+5, szSource, 6); //copy first 6 characters of
szSource to szDest, starting at szDest's 5th character
```

wcspbrk()

```
#include <string.h>
```

Scans strings for characters in specified character sets.

Example:

```
LPWSTR wszStr = TEXT("Test+String");
LPWSTR wszSearchChars = TEXT("+");

//locate the first occurence of any character in the wszSearchChars
list
LPWSTR pszFound = wcspbrk(wszStr, wszSearchChars);    //pszFound
should point to "+String"
```

wcsrchr()

```
#include <string.h>
```

(or wchar.h)

Finds a Unicode character in a Unicode string and returns a pointer to the first occurrence of that character in the string.

Example:

```
wcsrchr(szString, 'c');
```

wcsspn()

```
#include <string.h>
```

(or wchar.h)

Searches a Unicode string szToSearch for a Unicode substring szSub; returns an integer specifying the number of characters in szToSearch before one of the characters in szSub occurs.

Example:

```
WCHAR szToSearch[] = TEXT("789areyou?");
WCHAR szSub[] = TEXT("eyou?");
wcsspn (szToSearch, szSub); //result will be 5
```

wcsstr()

```
#include <string.h>
```

(or wchar.h)

Finds a Unicode substring szSub in a Unicode string szString; returns a pointer to the first occurrence of szSub in szString.

Example:

```
szFirstOccurrence = wcsstr(szString, szSub);
```

wcstod()

```
#include <string.h>
```

(or wchar.h)

Converts a string szNum to a (double) floating-point value; stops scanning at first non-numeric character and returns a pointer to this first non-numeric character as szStopAt.

Example:

```
fFloatVal = wcstod(szNum, &szStopAt);
```

wcstok()

```
#include <string.h>
```

(or wchar.h)

Parses a Unicode string by searching for single Unicode character delimiters in the string; returns a pointer to the next token.

Example:

```
WCHAR szStrToParse[] = TEXT("Windows CE version 2.11");
WCHAR* szToken = wcstok(szStrToParse, TEXT(" ")); //szToken will
point to "Windows"
szToken = wcstok(NULL, TEXT(" ")); //szToken will point to "CE"
//...etc.
```

wcstol()

```
#include <string.h>
```

(or wchar.h)

Converts a string szNum to a Long integer value; stops scanning at first non-numeric character and returns a pointer to this first non-numeric character as szStopAt.

Example:

```
fLongIntVal = wcstol(szNum, &szStopAt);
```

wcstombs()

```
#include <string.h>
```

(or wchar.h)

Converts a sequence of Unicode characters to a corresponding sequence of multibyte or ANSI characters.

Example:

```
wcstombs(szANSIString, szWideString, wcslen(szWideString) *
sizeof(WCHAR));
```

wcstoul()

```
#include <string.h>
```

(or wchar.h)

Converts a string szNum to a Long unsigned integer value; stops scanning at first non-numeric character and returns a pointer to this first non-numeric character as szStopAt.

Example:

```
fUIntVal = wcstoul(szNum, &szStopAt);
```

wsprintf()

```
#include <winbase.h>
```

A printf()-like function that "prints" the result to a string instead of a console.

Example:

```
wsprintf(szMsg, TEXT("%d:%d -%s"), 5024, 898, szTemp);
```

APPENDIX

B

The CE 2.0 API

This appendix contains a list of those API functions specific to Windows CE 2.0. The format that we will use to describe these functions is as follows:

Function Name

```
#include <Header file to be included>
```

Brief description

Example

Explanation of example

Unless otherwise explicitly stated, all strings are assumed to be Unicode-based or wide strings. If a function has a RAPI version that is named the same as the CE-based version, the CE-based header file will be given first, with `rapi.h` following on the next line to indicate that a RAPI version of that function (with that name) also exists.

AddAddressCard()

```
#include <addrstor.h>
```

One of the Contacts database functions; adds a new entry to the Contacts database.

Example:

```
AddAddressCard(&ac, &oidCard, &index);
```

Where:

- `ac` is an AddressCard holding the information to be added to the database.
- `oidCard` is a `CEOID`; the function will return the `CEOID` of the record after it has been added.
- `index` is an integer; the function will return the index of the record after it has been added.

The AddressCard structure is defined as follows:

```
typedef struct _AddressCard {
SYSTEMTIME stBirthday;
SYSTEMTIME stAnniversary;
TCHAR *pszBusinessFax;
TCHAR *pszCompany;
```

```
TCHAR *pszDepartment;
TCHAR *pszEmail;
TCHAR *pszMobilePhone;
TCHAR *pszOfficeLocation;
TCHAR *pszPager;
TCHAR *pszWorkPhone;
TCHAR *pszTitle;
TCHAR *pszHomePhone;
TCHAR *pszEmail2;
TCHAR *pszSpouse;
TCHAR *pszNotes;
TCHAR *pszEmail3;
TCHAR *pszHomePhone2;
TCHAR *pszHomeFax;
TCHAR *pszCarPhone;
TCHAR *pszAssistant;
TCHAR *pszAssistantPhone;
TCHAR *pszChildren;
TCHAR *pszCategory;
TCHAR *pszWebPage;
TCHAR *pszWorkPhone2;
TCHAR *pszNamePrefix;
TCHAR *pszGivenName;
TCHAR *pszMiddleName;
TCHAR *pszSurname;
TCHAR *pszGeneration;
TCHAR *pszHomeAddrStreet;
TCHAR *pszHomeAddrCity;
TCHAR *pszHomeAddrState;
TCHAR *pszHomeAddrPostalCode;
TCHAR *pszHomeAddrCountry;
TCHAR *pszOtherAddrStreet;
TCHAR *pszOtherAddrCity;
TCHAR *pszOtherAddrState;
TCHAR *pszOtherAddrPostalCode;
TCHAR *pszOtherAddrCountry;
TCHAR *pszOfficeAddrStreet;
TCHAR *pszOfficeAddrCity;
TCHAR *pszOfficeAddrState;
TCHAR *pszOfficeAddrPostalCode;
TCHAR *pszOfficeAddrCountry;
BYTE *rgbReserved[84]
} AddressCard;
```

BatteryNotifyOfTimeChange()

```
#include <winbase.h>
```

Notifies battery of system time change. Used by OS to recalculate battery elapsed time when user or an application changes system time.

Example:

```
SYSTEMTIME st;
FILETIME ft;
BOOL bAhead;
GetSystemTime(&st);
bAhead = TRUE;
st.wHour = st.wHour + 1; //daylight savings? spring ahead
SetSystemTime(&st);
memset(&st, "/0", sizeof(st));
st.wHour = 1;
SystemTimeToFileTime(&st, &ft)
BatteryNotifyOfTimeChange(bAhead, &ft);
```

Where:

- bAhead is a BOOL indicating whether the time is being set ahead (TRUE) or backward (FALSE).
- ft is a FILETIME 64-bit time structure.

TIP

It is almost always easier to use the SYSTEMTIME structure and then convert to a FILETIME, as opposed to attempting to directly manipulate the FILETIME structure. This is shown in the above example.

CeCheckPassword()

```
#include <winbase.h>
```

(or `rapi.h`)

Checks supplied password against system password.

Example:

```
TCHAR szPwd[MAX_PATH] = TEXT("IamTheWalrus");
CeCheckPassword(szPwd);
```

Where:

- szPwd is a string specifying the password to be checked.

CeClearUserNotification()

```
#include <notify.h>
```

Removes or clears a User Notification created with a prior call to CeSetUserNotification().

Example:

```
CeClearNotification(hNotification);
```

Where:

- hNotification is a handle to an existing notification.

CeCloseHandle()

```
#include <rapi.h>
```

RAPI version of the CloseHandle() function; closes an active handle to an object.

Example:

```
CeCloseHandle(hCeObject);
```

Where:

- hCeObject is an active handle to a CE system object such as a file.

TIP Although CeFindClose() is used to close a CeFindFirstFile() search, CeCloseHandle() should be used to close a CeFindFirstDatabase() search.

CeCopyFile()

```
#include <rapi.h>
```

RAPI version of the CopyFile() function; copies an existing file on the CE device to a new path/file on the CE device.

Example:

```
TCHAR szSource[MAX_PATH] = TEXT("\\source.txt");
TCHAR szDest[MAX_PATH] = TEXT("\\dest.txt");
bFailOnExist = FALSE;
CeCopyFile(szSource, szDest, bFailOnExist);
```

Where:

- `szSource` is the source file.
- `szDest` is the destination file.
- `bFailOnExist` is a `BOOL` indicating whether the function should fail if the destination file already exists.

CeCreateDatabase()

```
#include <winbase.h>
```

(or `rapi.h`)

Part of the CE database engine; creates a database.

Example:

```
TCHAR szName[MAX_PATH] = TEXT("MasterDB"); //set name of database
//set up sort orders
soSortOrders[SORT_ORDER_CODE].propid =
MAKELONG(CEVT_LPWSTR,PROP_CODE);
soSortOrders[SORT_ORDER_CODE].dwFlags= CEDB_SORT_GENERICORDER;
//Ascending (default) order
soSortOrders[SORT_ORDER_STATE].propid =
MAKELONG(CEVT_LPWSTR,PROP_STATE);
soSortOrders[SORT_ORDER_CODE].dwFlags= CEDB_SORT_GENERICORDER;
//Ascending (default) order
wCount = 2;  //two sort orders
oidObjId = CeCreateDatabase(szName, DB_ID, wCount, soSortOrders);
```

Where:

- `szName` is a string containing the name of the database.
- `DB_ID` is an integer identifying the database. The database ID is intended to be used as a unique ID by which applications may reference a database instead of using the database's name, usually a `#define` constant.

- wCount is a WORD specifying the number of sort orders (maximum is 4).
- soSortOrders is an array of SORTORDERSPEC structures used to specify the sort orders of the database.
- oidObjId is a CEOID. The function will return the CEOID of the database created.

The SORTORDERSPEC structure is defined as follows:

```
typedef struct _SORTORDERSPEC {
PEGPROPID propid;
DWORD dwFlags;
} SORTORDERSPEC;
```

CeCreateDirectory()

```
#include <rapi.h>
```

RAPI version of CreateDirectory() function; creates a directory on the CE device.

Example:

```
TCHAR szPath[MAX_PATH] = TEXT("\\My Documents\\My App");
CeCreateDirectory(szPath, NULL);
```

Where:

- szPath is a string specifying the directory to be created.

TIP

The second parameter is always NULL on CE; on 98/NT it would specify the security attributes of the directory.

CeCreateFile()

```
#include <rapi.h>
```

RAPI version of CreateFile() function; creates a file for reading or writing and returns an active handle to that file.

Example:

```
TCHAR szFileName[MAX_PATH] = TEXT("\\data.txt");
wAccess = GENERIC_WRITE;
wShare = 0;
```

```
dwCreate = CREATE_ALWAYS;
dwFlags = FILE_ATTRIBUTE_NORMAL;
hSrc = CeCreateFile(
szFileName,
wAccess,
wShare,
NULL,
dwCreate,
dwFlags,
NULL);
```

Where:

- `szFileName` is a string specifying the name of the file to open.
- `wAccess` is a `WORD` specifying the type of access (`GENERIC_READ` and/or `GENERIC_WRITE`).
- `wShare` is a `WORD` specifying under what conditions another application may open or modify the file while you are using it (`0` for no access, `FILE_SHARE_READ` to permit read access, and `FILE_SHARE_WRITE` to permit write access).
- `dwCreate` is a `DWORD` specifying whether you are attempting to open an existing file or create a file (`CREATE_ALWAYS` always creates the file, `OPEN_EXISTING` attempts to open an existing file, etc.).
- `dwFlags` is a `DWORD` specifying additional file attributes, such as the archive, hidden, or read-only flags.

TIP The fourth parameter specifies security attributes and the seventh specifies a template file. Both are always NULL on CE.

CeCreateProcess()

```
#include <rapi.h>
```

RAPI version of `CreateProcess()` function; launches a program on the CE device.

Example:

```
//launches Pocket Word
TCHAR szAppName[MAX_PATH] = TEXT("\\windows\\pword.exe");
TCHAR szCmdLine[MAX_PATH] = TEXT("\\My Documents\\AWordFile.doc");
dwCreate = 0;
CeCreateProcess(szAppName, szCmdLine, NULL, NULL, FALSE, dwCreate,
NULL, NULL, NULL, &pi);
```

Where:

- `szAppName` is the path and file name of the application to be started on the CE device.
- `szCmdLine` is a string specifying the command line to be passed to the application being started; can be NULL.
- `dwCreate` is a `DWORD` specifying creation options, such as whether the process is to be created suspended, and so on. In this example, we set `dwCreate` to `0` so that the process will be launched immediately.
- `pi` is a structure of type `PROCESS_INFORMATION`; used to return information about the started process to the calling application.

The `PROCESS_INFORMATION` structure is defined as follows:

```
typedef struct _PROCESS_INFORMATION {
HANDLE hProcess;
HANDLE hThread;
DWORD dwProcessId;
DWORD dwThreadId;
} PROCESS_INFORMATION;
```

CeDeleteDatabase()

```
#include <winbase.h>
```

(or `rapi.h`)

Deletes a database from the CE device.

Example:

```
CeDeleteDatabase(CeOid);
```

Where:

- `CeOid` is a `CEOID` of a valid, existing database.

WARNING Although `CeDeleteDatabase()` is sensitive to sharing violations (i.e., it will not delete the database if it is being actively used by another process), it does not check to ensure that the database is empty before deleting it. In other words, it is entirely possible to delete a database with perfectly good data in it.

CeDeleteFile()

```
#include <rapi.h>
```

RAPI version of `DeleteFile()` function; deletes a file from the CE device.

Example:

```
TCHAR szFileName[MAX_PATH] = TEXT("\\deleteme.txt");
CeDeleteFile(szFileName);
```

Where:

- `szFileName` is the name of the file to be deleted.

CeDeleteRecord()

```
#include <winbase.h>
```

(or `rapi.h`)

Deletes the specified record from the specified database.

Example:

```
CeDeleteRecord( hDatabase, oid );
```

Where:

- `hDatabase` is a valid handle to a database.
- `oid` is the `CEOID` of the record to be deleted.

CeFindAllDatabases()

```
#include <rapi.h>
```

RAPI function that returns a list of all databases of a certain type on the CE device.

Example:

```
dwType = 0;
LPCEDB_FIND_DATA pFindData;
wFlags = FAD_OID | FAD_FLAGS | FAD_NAME | FAD_TYPE FAD_NUM_RECORDS
| FAD_NUM_SORT_ORDER | FAD_SORT_SPECS;
CeFindAllDatabases( DbType, 0xFF, &cFound, &pFindData);
```

Where:

- dwType is a DWORD specifying the database type ID to search for; 0 retrieves all databases.
- wFlags is a WORD specifying what pieces of information the function should retrieve. (In the example above, all possible flags are retrieved; the same effect could be achieved by setting this parameter to 0xFF.)
- cFound is an integer used by the function to indicate the number of matching databases found.
- pFindData is a pointer to an array of cFindData elements, where each element is a CEDB_FIND_DATA structure.

The CEDB_FIND_DATA structure is defined as follows:

```
struct CEDB_FIND_DATA {
CEOID OidDb;
CEDBASEINFO DbInfo;
};
```

The CEDBASEINFO, which you'll notice is a member of the CEDB_FIND_DATA structure, is defined as follows:

```
typedef struct _CEDBASEINFO {
DWORD dwFlags;
WCHAR szDbaseName[CEDB_MAXDBASENAMELEN];
DWORD dwDbaseType;
WORD wNumRecords;
WORD wNumSortOrder;
DWORD dwSize;
FILETIME ftLastModified;
SORTORDERSPEC rgSortSpecs[CEDB_MAXSORTORDER];
} CEDBASEINFO;
```

CeFindAllFiles()

```
#include <rapi.h>
```

RAPI function that returns a list of all files on the CE device matching the specified criteria.

Example:

```
TCHAR szSearchPath[MAX_PATH] = TEXT("\\*.txt");
dwAttributes = FAF_ATTRIBUTES | FAF_NAME;
CeFindAllFiles(szSearchPath, dwAttributes, &cFound, &pFindData);
```

Where:

- `szSearchPath` is a string specifying the path and/or filename to search for; can contain wildcards.
- `dwAttributes` is a `DWORD` used to set searching options; can be used to filter certain file types and tell the function which pieces of information you want it to return to your program.
- `cFound` is an integer used by the function to return the number of matching files found.
- `pFindData` is a pointer to an array of `cFound` elements where each element is a `CE_FIND_DATA` structure.

The `CE_FIND_DATA` structure is defined as follows:

```
typedef struct _CE_FIND_DATA {
DWORD dwFileAttributes;
FILETIME ftCreationTime;
FILETIME ftLastAccessTime;
FILETIME ftLastWriteTime;
DWORD nFileSizeHigh;
DWORD nFileSizeLow;
DWORD dwOID;
WCHAR cFileName[MAX_PATH];
} CE_FIND_DATA;
```

CeFindClose()

```
#include <rapi.h>
```

RAPI version of `FindClose()` function; closes a search handle and frees memory associated with Find operation.

Example:

```
hFind = CeFindFirstFile( TEXT("\\*.exe"), &wfd);
do
{
    //do something with found files
}while(CeFindNextFile(hFind, &wfd));
CeFindClose(hFind);
```

Where:

- hFind is a handle to the search operation.

CeFindFirstDatabase()

```
#include <winbase.h>
```

(or rapi.h)

Finds the first database matching the criteria.

Example:

```
dwDatabaseType = 0;
hFind = CeFindFirstDatabase(dwDatabaseType);
do
{
    //...
} while (oid = CeFindNextDatabase( hFind ));
CloseHandle( hFind );
```

Where:

- dwDatabaseType is a DWORD specifying either the database type ID to search for, or 0 to retrieve all databases.

CeFindFirstFile()

```
#include <rapi.h>
```

RAPI version of FindFirstFile() function; finds first file matching specified criteria.

Example:

```
TCHAR szFileSpec[MAX_PATH] = TEXT("\\*.exe");
hFind = CeFindFirstFile(szFileSpec, &wfd);
```

```
do
{
    //do something with found files
}while(CeFindNextFile(hFind, &wfd));
CeFindClose(hFind);
```

Where:

- `szFileSpec` is a string specifying the filename to match; can contain wildcards.

CeFindNextDatabase()

```
#include <winbase.h>
```

(or `rapi.h`)

Finds the next database matching the criteria; used in conjunction with `CeFindFirstDatabase()`.

Example:

```
dwDatabaseType = 0;
hFind = CeFindFirstDatabase(dwDatabaseType);
do
{
    //...
} while (oid = CeFindNextDatabase( hFind ));
CloseHandle( hFind );
```

Where:

- `hFind` is a valid handle returned by `CeFindFirstDatabase()`.

CeFindNextFile()

```
#include <rapi.h>
```

RAPI version of `FindNextFile()` function; finds next file matching specified criteria.

Example:

```
TCHAR szFileSpec[MAX_PATH] = TEXT("\\*.exe");
hFind = CeFindFirstFile(szFileSpec, &wfd);
do
{
```

```
    //do something with found files
}while(CeFindNextFile(hFind, &wfd));
CeFindClose(hFind);
```

Where:

- szFileSpec is a string specifying the filename to match; can contain wildcards.

CeGetClassName()

```
#include <rapi.h>
```

Retrieves the class name of a given window handle.

Example:

```
GetClassName(hWnd, szClass, iLength);
```

Where:

- hWnd is the handle of the window whose class name is being requested.
- szClass is a string the function will use to store the class name of the window.
- iLength is the maximum length, in number of characters, of szClass.

CeGetDesktopDeviceCaps()

```
#include <rapi.h>
```

RAPI version of GetDeviceCaps() function; retrieves one of several possible pieces of information about the CE device.

Example:

```
iParamToRetrieve = HORZSIZE;
iWidth = CeGetDesktopDeviceCaps(iParamToRetrieve);
```

Where:

- iParamToRetrieve is an integer specifying one of several possible flags that tell the function which piece of information to retrieve. The value returned will be an integer containing the value requested. Some common flags specified are

  HORZSIZE Function retrieves the horizontal dimensions of the device's screen.

VERTSIZE Function retrieves the vertical dimensions of the device's screen.

HORZRES Function retrieves the horizontal resolution of the device's screen.

VERTRES Function retrieves the vertical resolution of the device's screen.

CeGetDeviceId()

```
#include <ceutil.h>
```

Retrieves the ID of the currently selected device. Although the CE devices registered with the system appear as "friendly names" in the Mobile Devices window, they are actually identified in the Desktop machine's registry via a unique ID assigned to each device when it first connects to the system. This function, then, is useful in enabling you to work with the registry to retrieve various settings and properties of the connected device.

Example:

```
iCurrentDeviceId = CeGetDeviceId();
```

Where:

- `iCurrentDeviceId` is an integer that will store the ID of the currently connected device.

TIP The device ID is a subkey of the `HKEY_CURRENT_USER\Software\Microsoft\Windows CE Services\Partners` key.

NOTE This is a Desktop-based function designed for working with registry entries related to CE Services. Unlike RAPI functions, you do *not* need to have a CE device connected in order to use the `ceutil` functions.

CeGetFileAttributes()

```
#include <rapi.h>
```

RAPI version of `GetFileAttributes()` function; retrieves attributes for a file, given the file name.

Example:

```
TCHAR szFile[MAX_PATH] = TEXT("\\windows\\pword.exe");
dwAttribs = CeGetFileAttributes(szFile);
```

Where:

- `szFile` is a string containing the name of the file whose attributes the function should retrieve.
- `dwAttribs` is a `DWORD` into which the function will return the file's attributes. In addition to the usual archive, read-only, etc. flags, Windows CE introduces two new file attributes:

  FILE_ATTRIBUTE_INROM A nonexecutable ROM file. Can be opened for read-only access.

  FILE_ATTRIBUTE_ROMMODULE An executable ROM file. Cannot be opened as a file, but must instead be executed (if an executable) or loaded (if a DLL).

CeGetFileSize()

```
#include <rapi.h>
```

RAPI version of `GetFileSize()` function; retrieves size of a file.

Example:

```
DWORD dwHigh;
dwSize = GetFileSize (hFile, &dwHigh);
```

Where:

- `hFile` is a handle to an open file.
- `dwSize` will receive the low 32 bits of the file's size.
- `dwHigh` will receive the high 32 bits of the file's size; may be NULL if files are known to have a size of 4GB or less.

CeGetFileTime()

```
#include <rapi.h>
```

RAPI version of `GetFileTime()` function; retrieves the created, last accessed, and last modified date-time stamps for a file.

Example:

```
CeGetFileTime(hFile, &ftCreate, &ftAccess, &ftModified);
```

Where:

- `hFile` is a handle to an open file.
- `ftCreate` is a `FILETIME` structure the function will use to return the created date-time.
- `ftAccess` is a `FILETIME` structure the function will use to return the created date-time.
- `ftModified` is a `FILETIME` structure the function will use to return the created date-time.

TIP

See `BatteryNotifyOfTimeChange()` for more information about the `FILETIME` structure and how to best use it.

WARNING

Only files stored on a Compact Flash card will report the same values for `ftModified` and `ftCreate`. Files stored in main storage (not a Compact Flash card) will always report all three `FILETIME` values as being the same as the last modified date `ftModified`.

CeGetLastError()

```
#include <rapi.h>
```

RAPI version of `GetLastError()` function; returns numeric value of last error generated due to error on the CE device itself.

Example:

```
hErrorCode = CeGetLastError();
```

Where:

- `hErrorCode` is an `HRESULT` value.

CeGetSelectedDeviceId()

```
#include <ceutil.h>
```

Related to `CeGetDeviceId()`; part of the RAPI-like `ceutil.h` functions.

Example:

```
dwDeviceId = CeGetSelectedDeviceId();
```

Where:

- `dwDeviceID` is a `DWORD` that will store the device ID returned by the function.

CeGetSpecialFolderPath()

```
#include <rapi.h>
```

RAPI version of the `GetSpecialFolderPath()` function; retrieves the true path of a system "special folder."

Example:

```
CeGetSpecialFolderPath(CSIDL_BITBUCKET, MAX_PATH, szPath);
```

Where:

- The first parameter specifies which path (i.e., special folder) to retrieve; in this case, the recycle bin.
- The second parameter specifies the length of `szPath`.
- `szPath` is a string that will hold the actual path of the special folder.

NOTE For more information on this function and its parameters, see Chapter 10.

CeGetStoreInformation()

```
#include <rapi.h>
```

RAPI version of GetStoreInformation() function; retrieves information about the CE object store (i.e., that part of RAM used as storage space).

Example:

```
CeGetStoreInformation(&siStoreInfo);
```

Where:

- siStoreInfo is a STORE_INFORMATION structure.

The STORE_INFORMATION structure is defined as follows:

```
typedef struct STORE_INFORMATION {
DWORD dwStoreSize;
DWORD dwFreeSize;
} STORE_INFORMATION;
```

CeGetSystemInfo()

```
#include <rapi.h>
```

RAPI version of GetSystemInfo(); retrieves generic operating system information about the CE device.

Example:

```
CeGetSystemInfo(&siSystemInfo);
```

Where:

- siSystemInfo is a SYSTEM_INFO structure.

The SYSTEM_INFO structure is defined as follows:

```
typedef struct _SYSTEM_INFO {
DWORD dwOemId;
DWORD dwPageSize;
LPVOID lpMinimumApplicationAddress;
LPVOID lpMaximumApplicationAddress;
DWORD dwActiveProcessorMask;
DWORD dwNumberOfProcessors;
DWORD dwProcessorType;
DWORD dwAllocationGranularity;
DWORD dwReserved;
} SYSTEM_INFO;
```

CeGetSystemMetrics()

```
#include <rapi.h>
```

RAPI version of GetSystemMetrics() function; retrieves sizes (in pixels) of certain graphical elements.

Example:

```
    iWidth = CeGetSystemMetrics(SM_CXSCREEN); //retrieves width of
screen
```

Where:

- iWidth is an integer that will store the value returned by the function. Common values for this parameter are

  SM_CXSCREEN Retrieves the width of the screen.

  SM_CYSCREEN Retrieves the height of the screen.

CeGetSystemPowerStatusEx()

```
#include <rapi.h>
```

RAPI version of GetSystemPowerStatusEx() function; retrieves information about the CE device's power source.

Example:

```
    bMostCurrentInfo = FALSE;
    CeGetSystemPowerStatusEx(&psPowerStatus, bMostCurrentInfo);
```

Where:

- psPowerStatus is a SYSTEM_POWER_STATUS_EX structure.
- bMostCurrentInfo is a BOOL value indicating whether the function should query the device driver directly for the most up-to-date information or whether it should use cached information that may be several seconds old.

The SYSTEM_POWER_STATUS_EX structure is defined as follows:

```
typedef struct _SYSTEM_POWER_STATUS_EX {
BYTE ACLineStatus;
BYTE BatteryFlag;
BYTE BatteryLifePercent;
BYTE Reserved1;
```

```
DWORD BatteryLifeTime;
DWORD BatteryFullLifeTime;
BYTE Reserved2;
BYTE BackupBatteryFlag;
BYTE BackupBatteryLifePercent;
BYTE Reserved3;
DWORD BackupBatteryLifeTime;
DWORD BackupBatteryFullLifeTime;
} SYSTEM_POWER_STATUS_EX;
```

CeGetTempPath()

```
#include <rapi.h>
```

Retrieves the CE device's TEMP path.

Example:

```
CeGetTempPath(dwStrLen, szPath);
```

Where:

- dwStrLen is a DWORD specifying the length of szPath.
- szPath is a string the function will use to return the TEMP path.

CeGetUserNotificationPreferences()

```
#include <notify.h>
```

Launches the Notification Options dialog with the options you set. The dialog allows the user to set options about how they'd like a notification to appear. The user's changes to the notification options are then returned via the structure passed as the second parameter.

Example:

```
unNotify.ActionFlags =  PUN_LED | PUN_DIALOG;
unNotify.pwszDialogTitle = TEXT("Notification!");
unNotify.pwszDialogText = TEXT("Your notification.");
CeGetUserNotificationPreferences(hWnd, &unNotify);
```

Where:

- hWnd is the handle of the calling application.

- unNotify is a CE_USER_NOTIFICATION structure. (PUN_LED specifies that the LED on the device should be blinked; PUN_DIALOG specifies that the user should be notified via a dialog box.)

The CE_USER_NOTIFICATION structure is defined as follows:

```
typedef struct UserNotificationType {
DWORD ActionFlags;
TCHAR *pwszDialogTitle;
TCHAR *pwszDialogText;
TCHAR *pwszSound;
DWORD nMaxSound;
DWORD dwReserved;
} CE_USER_NOTIFICATION
```

WARNING Calling the CeGetUserNotificationPreferences() function and displaying the Options dialog does not actually create or set the actual notification. The application must still call CeSetUserNotification() itself.

NOTE Giving the user the option to alter the notification settings is purely optional; it's possible to create a notification using any combination of preferences and settings without ever consulting the user.

CeGetVersionEx()

```
#include <rapi.h>
```

RAPI version of GetVersionEx() function; returns version information.

Example:

```
CeGetVersionEx(&viOSVerInfo);
```

Where:

- viOSVerInfo is an OSVERSIONINFO structure.

The OSVERSIONINFO structure is defined as follows:

```
typedef struct _OSVERSIONINFO{
DWORD dwOSVersionInfoSize;
DWORD dwMajorVersion;
```

```
DWORD dwMinorVersion;
DWORD dwBuildNumber;
DWORD dwPlatformId;
TCHAR szCSDVersion[128];
) OSVERSIONINFO;
```

CeGetWindow()

```
#include <rapi.h>
```

RAPI version of `GetWindow()` function; returns the handle of a window matching the criteria specified.

Example:

```
uWndType = GW_HWNDFIRST;
hMatchingWnd = CeGetWindow(hWnd, uWndType);
```

Where:

- `hWnd` is the handle to a window, usually that of the calling application.
- `uWndType` is a `UINT` specifying the criteria of the window to search for, in this case, the window at the top of the Z-order.
- `hMatchingWnd` is a handle to the window matching the criteria, or NULL if no window matches.

CeGetWindowLong()

```
#include <rapi.h>
```

RAPI version of `GetWindowLong()` function; retrieves extended information about the window specified.

Example:

```
iVal = GWL_WNDPROC;
DefEditProc = GetWindowLong(hWnd, iVal);
```

Where:

- `hWnd` is the handle to the window whose information the function should retrieve.
- `iVal` is an integer specifying the piece of information the function should retrieve, in this case, the address of the `hWnd`'s `WndProc()` function.

CeGetWindowText()

```
#include <rapi.h>
```

RAPI version of `GetWindowText()` function; retrieves the caption or text displayed by the window specified.

Example:

```
GetWindowText(hWnd, szText, iSize);
```

Where:

- `hWnd` is a handle to a window.
- `szText` is a string that will receive the window's text.
- `iSize` is the size or maximum length of `szText`.

CeGlobalMemoryStatus()

```
#include <rapi.h>
```

RAPI version of `GlobalMemoryStatus()` function; retrieves information about memory of the CE device.

Example:

```
CeGlobalMemoryStatus(&msMemStatus);
```

Where:

- `msMemStatus` is a `MEMORYSTATUS` structure.

The `MEMORYSTATUS` structure is defined as follows:

```
typedef struct _MEMORYSTATUS {
DWORD dwLength;
DWORD dwMemoryLoad;
DWORD dwTotalPhys;
DWORD dwAvailPhys;
DWORD dwTotalPageFile;
DWORD dwAvailPageFile;
DWORD dwTotalVirtual;
DWORD dwAvailVirtual;
} MEMORYSTATUS
```

CeHandleAppNotifications()

```
#include <notify.h>
```

Marks all "triggered" notifications created by the calling application as handled.

Example:

```
CeHandleAppNotifications(szAppName);
```

Where:

- szAppName is a string containing the name of the application, as it was specified in calls to CeSetUserNotification().

CeMoveFile()

```
#include <rapi.h>
```

RAPI version of MoveFile() function; moves (i.e., renames) a file or directory on the CE device.

Example:

```
CeMoveFile(szOldName, szNewName);
```

Where:

- szOldName is a string specifying the current name of the file or directory.
- szNewName is a string specifying the new name of the file or directory.

CeOidGetInfo()

```
#include <winbase.h>
```

(or rapi.h)

Retrieves information about a CEOID; can be used to identify the type of object specified, etc.

Example:

```
CeOidGetInfo(oid, &oiOidInfo) ;
```

Where:

- `oid` is a `CEOID` whose properties the function is to retrieve.
- `oiOidInfo` is a `CEOIDINFO` structure.

The `CEOIDINFO` structure is defined as follows:

```
typedef struct _CEOIDINFO {
WORD wObjType;
DWORD dwSize;
WORD wPad;
union {
CEFILEINFO infFile;
CEDIRINFO infDirectory;
CEDBASEINFO infDatabase;
CERECORDINFO infRecord;
};
} CEOIDINFO;
```

CeOpenDatabase()

```
#include <winbase.h>
```

(or `rapi.h`)

Part of the CE database engine; opens a CE database.

Example:

```
oid = 0;
TCHAR szDBName[MAX_PATH] = TEXT("My DB");
m_Handle = CeOpenDatabase(&oid, szDBName, 0, 0, NULL);
```

Where:

- `oid` is a `CEOID` (can be `0`, if the `CEOID` of the database is unknown, but the `szDBName` is supplied. In that case, the `CEOID` of the database will be written back to `oid`).
- `szDBName` is a string containing the name of the database to be opened.

TIP

The remaining parameters are explored in greater detail in Chapter 4.

CeRapiFreeBuffer()

```
#include <rapi.h>
```

Frees memory allocated by calls to RAPI functions such as `CeFindAllDatabases()`, `CeFindAllFiles()`, or `CeReadRecordProps()`.

Example:

```
CeFindAllDatabases(DbType, 0xFF, &cFound, &pFindData);
//do something with info...then, when done...
CeRapiFreeBuffer(pFindData);
```

Where:

- `pFindData` is a pointer to an array created by a call to `CeFindDatabases()`.

CeRapiGetError()

```
#include <rapi.h>
```

Returns an error value if a call to a RAPI function failed due to a RAPI-related problem (e.g., the device was suddenly disconnected) as opposed to a CE-related problem (e.g., one of the parameters contained an illegal value).

Example:

```
if (CeRapiGetError() != NO_ERROR)
{
    //Function failed due to RAPI-related problem
}
```

NOTE This function takes no parameters and returns either `NO_ERROR` or an error-specific code.

CeRapiInit()

```
#include <rapi.h>
```

Ensures a device is connected and initializes RAPI library.

Example:

```
if ( CeRapiInit() != E_FAIL )
{
    //Connected to device successfully!
}
```

NOTE This function takes no parameters and returns E_FAIL if unsuccessful.

CeRapiInitEx()

```
#include <rapi.h>
```

"Connects" to device, initializes RAPI library, and returns an event handle.

Example:

```
riRapiInit.cbSize = sizeof(RAPIINIT);
CeRapiInitEx(&riRapiInit);
```

Where:

- `riRapiInit` is a `RAPIINIT` structure, which the function uses to return the event handle.

The `RAPIINIT` structure is defined as follows:

```
typedef struct _RAPIINIT {
DWORD cbSize;
HANDLE heRapiInit;
HANDLE hrRapiInit;
} RAPIINIT;
```

CeRapiInvoke()

```
#include <rapi.h>
```

Executes a function in a DLL located on the CE device.

Example:

```
WCHAR szDllName[MAX_PATH] = TEXT("\\Windows\\mydll.dll");
WCHAR szTmp[MAX_PATH];
WCHAR szFxn[MAX_PATH];
WCHAR szInput[MAX_PATH];
LPTSTR szOut = NULL;
BOOL fStream = FALSE;
IRAPIStream *pStream = NULL;
DWORD dwInput = 0;
DWORD dwOut = MAX_PATH;
TCHAR sz[MAX_PATH];
```

```
    //...
    CeRapiInit();
    //...
    HRESULT  hr = CeRapiInvoke(szDllName, szFxn, dwInput, (BYTE *)
szInput, &dwOut, (BYTE **)&szOut, fStream ? &pStream : NULL, NULL);
```

Where:

- `szDllName` is a string specifying the name of the DLL.
- `szFxn` is a string specifying the name of the function.
- `dwInput` is a `DWORD` specifying the length in bytes of `szInput`.
- `szInput` is a string specifying the input to the function.
- `dwOut` is a `DWORD` that will receive the length in bytes of `szOut`.
- `szOut` is an ANSI string that will receive the output data.
- `pStream` is an `IRAPIStream` interface.

NOTE The last parameter of this function is always NULL.

CeRapiUninit()

```
#include <rapi.h>
```

"Disconnects" from device and uninitializes RAPI library.

Example:

```
CeRapiUninit();
```

NOTE This function takes no parameters and returns `E_FAIL` only if the RAPI libraries were not initialized to begin with.

CeReadFile()

```
#include <rapi.h>
```

RAPI version of `ReadFile()` function; reads data from a file opened with `CreateFile()`.

Example:

```
CeReadFile(hFile, &szData, dwBytesToRead, &dwBytesRead, NULL);
```

Where:

- `hFile` is a handle to a file.
- `szData` is a string or buffer to hold the data being read.
- `dwBytesToRead` is a `DWORD` specifying number of bytes to read.
- `dwBytesRead` is a `DWORD` used by the function to indicate number of bytes actually read.

TIP The last parameter of `CeReadFile()` is always NULL.

CeReadRecordProps()

```
#include <winbase.h>
```

(or `rapi.h`)

Reads the properties (fields) of a record in a CE database.

Example:

```
objId = CeReadRecordProps(hDB,CEDB_ALLOWREALLOC,
&cProps,  NULL, (LPBYTE *)&pBuf, &cbBuf);
```

Where:

- `hDB` is a handle to an open database.
- `CEDB_ALLOWREALLOC` is a flag that tells the function it can reallocate the buffer passed as the fifth parameter if necessary; can be 0 for no reallocation.
- `cProps` is an integer used by the function to return the number of properties read.
- NULL specifies that you want the function to read all of the record's properties.
- `pBuf` is a pointer to an array of `CEPROPVAL` structures.
- `cbBuf` is a `DWORD` specifying the size of `pBuf` in bytes; useful mostly if `CEDB_ALLOWREALLOC` is not specified.

The CEPROPVAL structure is defined as follows:

```
typedef struct _CEPROPVAL {
CEPROPID propid;
WORD wLenData;
WORD wFlags;
CEVALUNION val;
} CEPROPVAL;
```

CeRegCloseKey()

```
#include <rapi.h>
```

RAPI version of `RegCloseKey()` function; closes handle to a specified registry key on the CE device.

Example:

```
CeRegCloseKey (hKey);
```

Where:

- `hKey` is a handle to a registry key.

CeRegCreateKeyEx()

```
#include <rapi.h>
```

RAPI version of `RegCreateKeyEx()` function; creates a registry key on the CE device.

Example:

```
HKEY  hKeyResult;
DWORD dwCreateStatus;
DWORD dwType;
DWORD dwBytes = 0;
TCHAR szKeyName[MAX_PATH] = TEXT("Preferences");
CeRegCreateKeyEx(hKey, szKeyName, 0, NULL, 0, 0 NULL, &hKeyResult,
&dwCreateStatus);
```

Where:

- hKey is a handle to the key under which the new key will be created. This parameter can be a handle to a key or any of the following predefined constants:
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
- szKeyName is the name of the new key.
- hKeyResult is a handle to receive the handle of the newly created key.
- dwCreateStatus is a DWORD used by the function to indicate whether the key was created (REG_CREATED_NEW_KEY) or whether it already existed (REG_OPENED_EXISTING_KEY).

CeRegDeleteKey()

```
#include <rapi.h>
```

RAPI version of RegDeleteKey(); deletes the specified key and any subkeys from the registry.

Example:

```
TCHAR szKeyName[MAX_PATH] = TEXT("Preferences");
//delete key created in example for CeRegCreateKeyEx() above
CeRegDeleteKey(HKEY_CURRENT_USER, szKeyName);
```

Where:

- HKEY_CURRENT_USER is a predefined constant value. This parameter can be a handle to a key or any of the following predefined constants:
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
- szKeyName is a string specifying the name of the key to be deleted.

CeRegDeleteValue()

```
#include <rapi.h>
```

RAPI version of `RegDeleteValue()` function; deletes the specified value from the registry of the CE device.

Example:

```
TCHAR szValName[MAX_PATH] = TEXT("value2");
LONG CeRegDeleteValue(hKey, szValName);
```

Where:

- `hKey` is a handle to a registry key. This parameter can be a handle to a key or any of the following predefined constants:
  - `HKEY_CLASSES_ROOT`
  - `HKEY_CURRENT_USER`
  - `HKEY_LOCAL_MACHINE`
  - `HKEY_USERS`
- `szValName` is a string specifying the value to be deleted.

CeRegEnumKeyEx()

```
#include <rapi.h>
```

RAPI version of `RegEnumKeyEx()` function; lists subkeys of the specified registry key on the CE device.

Example:

```
    while (ERROR_NO_MORE_ITEMS != CeRegEnumKeyEx(hKey, iIndex,
szKeyName, &cbKeyName, NULL, szClassName, &cbClassName, NULL )
    {
        //do something...
        iIndex++; //always increment index value
        cbClassName = 0; //these values must be reset each time
        cbKeyName = 0;
    }
```

Where:

- hKey is a handle to a registry key. This parameter can be a handle to a key or any of the following predefined constants:
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
- iIndex is an integer specifying the index of the subkey to be enumerated.
- szKeyName is a string to receive the name of the subkey.
- cbKeyName is an integer specifying the maximum length of szKeyName in bytes.
- szClassName is a string to receive the class name of the subkey.
- cbClassName is an integer specifying the maximum length of szClassName in bytes.

TIP The fifth and eighth parameters of this function are always NULL.

CeRegEnumValue()

```
#include <rapi.h>
```

RAPI version of RegEnumValue(); lists values belonging to the specified key.

Example:

```
    retValue = CeRegEnumValue (hKey, dwIndex, szValName, &dwValLen,
NULL, &dwValType, (LPBYTE)szData, &dwDataLen);
```

Where:

- hKey is a handle to a key. This parameter can be a handle to a key or any of the following predefined constants:
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS

- dwIndex is a DWORD specifying the index of the value to retrieve; this value should be 0 on the first call to the function and incremented before each additional call.
- szValName is a string used by the function to return the name of the value.
- dwValLen is a DWORD specifying the size of the szValName buffer.
- dwValType is a DWORD used by the function to return the type of the value. The most common values returned to this parameter are

  REG_BINARY Binary data

  REG_DWORD A DWORD

  REG_SZ A string
- szData is a string or byte array used to hold the data being returned.
- dwDataLen specifies the length of the szData buffer in bytes.

NOTE The fifth parameter of this function is always NULL.

CeRegOpenKeyEx()

```
#include <rapi.h>
```

RAPI version of RegOpenKeyEx(); opens the specified registry key.

Example:

```
CeRegOpenKeyEx (hKey, szKeyName, 0, KEY_READ, &hSubKey);
```

Where:

- hKey is a handle to a key. This parameter can be a handle to a key or any of the following predefined constants:
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS

- szKeyName specifies the name of the key to be opened. This key must be a subkey of the key specified by hKey.
- KEY_READ specifies the type of access/operation we'll be performing. Some common values are
  - KEY_READ
  - KEY_WRITE
  - KEY_ALL_ACCESS
- hSubKey is a handle to the key specified by szKeyName, if the function was successful.

CeRegQueryInfoKey()

RAPI version of RegQueryInfoKey(); retrieves information about the specified registry key.

Example:

```
CeRegQueryInfoKey(hKey, szClassName, &dwClassLen, NULL,
&dwSubKeyCount, &dwMaxSubKeyLen, &dwMaxClassLen, &dwValueCount,
&dwMaxValueName, NULL, NULL, NULL);
```

Where:

- hKey is a handle to a key. This parameter can be a handle to a key or any of the following predefined constants:
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
- szClassName is a string the function uses to return the class name of the key.
- dwClassLen is the size of the szClassName buffer.
- dwSubKeys is a DWORD the function uses to return the number of subkeys.
- dwMaxSubKeyLen is a DWORD the function uses to return the length of the longest subkey.

- dwMaxClassLen is a DWORD the function uses to return the length of the longest class name of a subkey.
- dwValueCount is a DWORD the function uses to return the number of values belonging to the key.
- dwMaxValueLen is a DWORD the function uses to return the length of the longest value name.

NOTE The fourth, tenth, eleventh, and twelfth parameters of this function are always NULL.

CeRegQueryValueEx()

```
#include <rapi.h>
```

RAPI version of RegQueryValueEx(); retrieves data and information about a specified value.

Example:

```
RegQueryValueEx (hKey, szValName, NULL, &dwValType, (LPBYTE)szData,
&dwDataLen);
```

Where:

- hKey is a handle to a key. This parameter can be a handle to a key or any of the following predefined constants:
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
- szValName is a string specifying the name of the value to retrieve.
- dwValType is a DWORD used by the function to return the type of the value. The most common values returned to this parameter are

  REG_BINARY Binary data

  REG_DWORD A DWORD

  REG_SZ A string

- szData is a string or byte array used to hold the data being returned.
- dwDataLen specifies the length of the szData buffer in bytes.

NOTE The third parameter of this function is always NULL.

CeRegSetValueEx()

```
#include <rapi.h>
```

RAPI version of RegSetValueEx(); writes data to the specified registry value.

Example:

```
CeRegSetValueEx(hKey, szValName, NULL, dwValType, (LPBYTE)szData,
dwDataLen);
```

Where:

- hKey is a handle to a key. This parameter can be a handle to a key or any of the following predefined constants:
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
- szValName is a string specifying the name of the value to retrieve.
- dwValType is a DWORD used by the function to return the type of the value. The most common values returned to this parameter are

  REG_BINARY Binary data

  REG_DWORD A DWORD

  REG_SZ A string
- szData is a string or byte array used to hold the data being returned.
- dwDataLen specifies the length of the szData buffer in bytes.

NOTE The third parameter of this function is always NULL.

CeRemoveDirectory()

```
#include <rapi.h>
```

RAPI version of `RemoveDirectory()`; deletes a directory. The directory must be empty before it can be removed.

Example:

```
TCHAR szDir[MAX_PATH] = TEXT("\\AppTempDir");
CeRemoveDirectory(szDir);
```

Where:

- `szDir` is the name of the directory to remove.

CeRunAppAtEvent()

```
#include <notify.h>
```

Runs the specified application when the specified system-level event occurs; similar to `CeRunAppAtTime()`.

Example:

```
TCHAR szAppName[MAX_PATH] = TEXT("\\windows\\pword.exe");
CeRunAppAtEvent(szAppName, NOTIFICATION_EVENT_TIME_CHANGE);
```

Where:

- `szAppName` is a string specifying the application to run.
- `NOTIFICATION_EVENT_TIME_CHANGE` is a predefined constant. Possible values for this second parameter are

  NOTIFICATION_EVENT_SYNC_END When the device is finished syncing with the Desktop.

  NOTIFICATION_EVENT_DEVICE_CHANGE When a PCMCIA card is removed or inserted.

  NOTIFICATION_EVENT_RS232_DETECTED When a serial port connection is made.

NOTIFICATION_EVENT_TIME_CHANGE When the system time has been changed.

NOTIFICATION_EVENT_RESTORE_END When a device data restore operation is completed.

To remove a previously set notification, the value to pass for the second parameter is

NOTIFICATION_EVENT_NONE Removes run-at settings for the specified application.

CeRunAppAtTime()

```
#include <notify.h>
```

Runs the specified application at the specified time; similar to `CeRunAppAtEvent()`.

Example:

```
TCHAR szAppName[MAX_PATH] = TEXT("\\windows\\pword.exe");
CeRunAppAtTime(szAppName, &stSystemTime);
```

Where:

- `szAppName` is a string specifying the application to run.
- `stSystemTime` is either a pointer to a SYSTEMTIME structure or NULL. If NULL, any previously set run-at-time events for the specified application will be cleared.

> For more information on working with the `SYSTEMTIME` structure, see `BatteryNotifyOfTimeChange()` earlier in this appendix.

CeSeekDatabase()

```
#include <winbase.h>
```

(or `rapi.h`)

Part of the CE database engine; serves as a way to both search the database and manipulate the current record pointer (seek).

Example:

```
CeSeekDatabase(hDb, dwSeekType, dwValue, &dwIndex);
```

Where:

- `hDb` is a handle to an open database.
- `dwSeekType` specifies the type of search/seek to perform.
- `dwValue` is the value you're searching for or is a numeric value specifying how the current record pointer should be moved.
- `dwIndex` is a `DWORD` that the CE Database Engine uses to return the number of records from the start of the database to the record that was found.

NOTE This function, and the dwSeekType parameter in particular, is fully documented in Chapter 4.

CeSetDatabaseInfo()

```
#include <winbase.h>
```

(or `rapi.h`)

Part of the CE Database Engine; sets various properties of the database.

Example:

```
CeSetDatabaseInfo(oid, &dbiDatabaseInfo);
```

Where:

- `oid` is the `CEOID` of the database whose properties are to be changed.
- `dbiDatabaseInfo` is a `CEDBASEINFO` structure.

The `CEDBASEINFO` is defined as

```
typedef struct _CEDBASEINFO {
DWORD dwFlags;
WCHAR szDbaseName[CEDB_MAXDBASENAMELEN];
DWORD dwDbaseType;
WORD wNumRecords;
WORD wNumSortOrder;
DWORD dwSize;
```

```
FILETIME ftLastModified;
SORTORDERSPEC rgSortSpecs[CEDB_MAXSORTORDER];
} CEDBASEINFO;
```

CeSetEndOfFile()

```
#include <rapi.h>
```

RAPI version of `SetEndOfFile()`; marks current position as the end of the file.

Example:

```
CeSetEndOfFile(hFile);
```

Where:

- `hFile` is a handle to an open file on the CE device; you must have write access to the file in order to successfully set the EOF of that file.

CeSetFileAttributes()

```
#include <rapi.h>
```

RAPI version of `SetFileAttributes()`; sets attributes of a file on the CE device.

Example:

```
TCHAR szFileName[MAX_PATH] = TEXT("\\myappdata.txt");
dwFileAttributes = FILE_ATTRIBUTE_NORMAL;
CeSetFileAttributes(szFileName, dwFileAttributes);
```

Where:

- `szFileName` is the name of the file whose attributes you want to change.
- `dwFileAttributes` is a DWORD specifying the attributes to set. It can be any combination of the following values:

  FILE_ATTRIBUTE_ARCHIVE Sets the archive flag.

  FILE_ATTRIBUTE_HIDDEN Sets the hidden flag.

  FILE_ATTRIBUTE_NORMAL Clears all other attributes; cannot be combined with other attributes.

  FILE_ATTRIBUTE_OFFLINE File has been moved to offline storage.

  FILE_ATTRIBUTE_READONLY Sets the read-only flag.

FILE_ATTRIBUTE_SYSTEM Sets the system flag.

FILE_ATTRIBUTE_TEMPORARY Marks the file as being for temporary use only, which can improve access time. It is still the responsibility of the application to remove the temporary file when finished with it.

CeSetFilePointer()

```
#include <rapi.h>
```

RAPI version of `SetFilePointer()`; advances or reverses the file pointer of an open file.

Example:

```
CeSetFilePointer(hFile, 3000, NULL, FILE_BEGIN);
```

Where:

- `hFile` is a handle to an open file.
- 3000 is the number of bytes to move.
- `FILE_BEGIN` is the point that the file pointer will move relative to. In other words, in the above example, the file pointer will be positioned 3000 bytes from the beginning of the file. Other possible values for this parameter are

  FILE_CURRENT Distance to move is relative to the current position of the file pointer.

  FILE_END Distance to move is relative to the EOF.

NOTE The third parameter can either be NULL, as in the example here, or it can serve as the high-order `DWORD` if the distance to move is a 64-bit value. In that case, the second parameter would be the low-order `DWORD`.

CeSetFileTime()

```
#include <rapi.h>
```

RAPI version of `SetFileTime()`; sets the creation, last accessed, and/or last modified date-time stamps of the specified file.

Example:

```
SYSTEMTIME st;
FILETIME ftCreate, ftAccess, ftModify;
BOOL bAhead;
GetSystemTime(&st);
//set all three dates equal to current date-time stamp
SystemTimeToFileTime(&st, &ftCreate);
SystemTimeToFileTime(&st, &ftAccess);
SystemTimeToFileTime(&st, &ftModify);
CeSetFileTime(hFile, &ftCreate, &ftAccess, &ftModify);
```

Where:

- `hFile` is a handle to an open file.
- `ftCreate` is a `FILETIME` structure specifying the new creation date-time stamp of the file.
- `ftAccess` is a `FILETIME` structure specifying the new last accessed date-time stamp of the file.
- `ftModify` is a `FILETIME` structure specifying the new last modified date-time stamp of the file.

CeSetUserNotification()

```
#include <notify.h>
```

Creates or updates a user notification.

Example:

```
BOOL ret;
SYSTEMTIME stTime, stFuture;
CE_USER_NOTIFICATION unNotify;
HANDLE hNotification;
TCHAR szAppFullPath[MAX_PATH] = TEXT("\\Windows\\Start Menu\\
helloworld.exe");
GetLocalTime(&stTime);
stFuture = stTime;
stFuture.wMinute = stFuture.wMinute + 1;
unNotify.ActionFlags =  PUN_LED | PUN_DIALOG;
unNotify.pwszDialogTitle = TEXT("Notification!");
unNotify.pwszDialogText = TEXT("Your notification is here!");
```

```
    hNotification = NULL;
    hNotification = CeSetUserNotification(hNotification, szAppFullPath,
&stFuture, &unNotify);
```

Where:

- `hNotification` is NULL to create a notification (as in this example) or the handle to an existing notification.
- `szAppFullPath` is a string specifying the full path to the executable creating the notification. This is the application whose icon will appear in the notification dialog box and the application that will be executed if the user clicks the Open button on the notification dialog.
- `stFuture` is a `SYSTEMTIME` structure specifying the date-time stamp at which the notification is to occur.
- `unNotify` is a `CE_USER_NOTIFICATION` structure, specifying various notification options.

The `CE_USER_NOTIFICATION` structure is defined as follows:

```
typedef struct UserNotificationType {
DWORD ActionFlags;
TCHAR *pwszDialogTitle;
TCHAR *pwszDialogText;
TCHAR *pwszSound;
DWORD nMaxSound;
DWORD dwReserved;
} CE_USER_NOTIFICATION
```

CeSHCreateShortcut()

```
#include <rapi.h>
```

RAPI version of `SHCreateShortcut()`; creates a shortcut.

Example:

```
TCHAR szShortcutFile[MAX_PATH];
TCHAR szActualExe[MAX_PATH];
szShortcutFile = TEXT("\\MyApp.lnk");
szActualExe = TEXT("\\Windows\\MyApp.exe");
CeSHCreateShortcut(szShortcutFile, szActualExe);
```

Where:

- szShortcutFile is a string specifying the full path of the shortcut file.
- szActualExe is a string specifying the full path to the actual executable file.

CeSHGetShortcutTarget()

RAPI version of SHGetShortcutTarget(); retrieves the path to the target executable of the specified shortcut.

Example:

```
TCHAR szShortcutFile[MAX_PATH];
TCHAR szActualExe[MAX_PATH];
int iActualExeLen = MAX_PATH;
szShortcutFile = TEXT("\\MyApp.lnk");
CeSHGetShortcutTarget(szShortcutFile, szActualExe, iActualExeLen);
```

Where:

- szShortcutFile is a string specifying the full path of the shortcut file.
- szActualExe is a string the function uses to return the full path to the actual executable file.
- iActualExeLen is an integer specifying the maximum possible length of szActualExe.

CeWriteFile()

```
#include <rapi.h>
```

RAPI version of WriteFile(); writes data to an open file.

Example:

```
CeWriteFile(hFile, szBuf, dwBufSize, &dwNumWritten, NULL));
```

Where:

- hFile is a handle to an open file.
- szBuf is a buffer of some kind.
- dwBufSize is a DWORD specifying the size of szBuf in bytes.
- dwNumWritten is a DWORD the function uses to return the number of bytes written to the file.

WARNING When using `CeWriteFile()` to write Unicode data to the file, remember that `dwBufSize` is the number of bytes—not characters—in the buffer. When writing Unicode text, you must remember to double the number of characters to get the number of bytes.

CeWriteRecordProps()

```
#include <winbase.h>
```

(or `rapi.h`)

Part of the CE Database Engine; creates or modifies a record in a database.

Example:

```
CEOID oidRecord;
WORD wPropCount;
CEPROPVAL pvPropVals[2];
pvPropVals[0].propid = MAKELONG(CEVT_LPWSTR, PROP_CODE);
pvPropVals[0].wFlags = 0;
pvPropVals[0].val.lpwstr = szCode;
pvPropVals[1].propid = MAKELONG(CEVT_LPWSTR, PROP_STATE);
pvPropVals[1].wFlags = 0;
pvPropVals[1].val.lpwstr = szState;
wPropCount = 2;
//...
//get a value for the record's oid & store in oidRecord
//...
CeWriteRecordProps(hDb, oidRecord, wPropCount, pvPropVals);
```

Where:

- `hDb` is a handle to an open database.
- `oidRecord` is the `CEOID` of the record being modified; if adding a new record, this value should be set to 0.
- `wPropCount` is a `WORD` indicating the number of properties being written to.
- `pvPropVals` is an array of `CEPROPVAL` structures.

The CEPROPVAL structure is defined as follows:

```
typedef struct _CEPROPVAL {
CEPROPID propid;
WORD wLenData;
WORD wFlags;
CEVALUNION val;
} CEPROPVAL;
typedef CEPROPVAL;
```

CloseAddressBook()

```
#include <addrstor.h>
```

Closes the Contacts database; part of the AddressBook family of functions.

Example:

```
CloseAddressBook();
```

NOTE This function takes no parameters and does not return a value.

CommandBands_AddAdornments()

```
#include <commctrl.h>
```

Adds the Close (X) button to the CommandBands control. Can also add Help (?) and/or OK buttons. These buttons are considered separate *adornments bands*.

Example:

```
DWORD dwFlags = CMDBAR_HELP | CMDBAR_OK;
REBARBANDINFO rbiInfo;
rbiInfo.cbSize = sizeof (REBARBANDINFO);
rbiInfo.fMask = RBBIM_ID;
rbiInfo.wID = 300;
CommandBands_AddAdornments(hBands, hInstance, dwFlags, &rbiInfo);
```

Where:

- `hBands` is a handle to the window's CommandBands control.
- `hInstance` is a handle to the instance of the application.

- `dwFlags` is a `DWORD` specifying one or more of the following values:

  CMDBAR_HELP Add the Help (?) button.

  CMDBAR_OK Add the OK button.

- `rbiInfo` is optionally NULL or a `REBARBANDINFO` structure, which can be used to override the default ID for the adornments band. Usually, the ID of the adornments band is set to `0xFFFFFFFF`; the example above sets it to 300.

The `REBARBANDINFO` structure is defined as follows:

```
typedef struct tagREBARBANDINFO{
UINT cbSize;
UINT fMask;
UINT fStyle;
COLORREF clrFore;
COLORREF clrBack;
LPTSTR lpText;
UINT cch;
int iImage;
HWND hwndChild;
UINT cxMinChild;
UINT cyMinChild;
UINT cx;
HBITMAP hbmBack;
UINT wID;
UINT cyChild;
UINT cyMaxChild;
UINT cyIntegral;
UINT cxIdeal;
LPARAM lParam;
} REBARBANDINFO;
```

CommandBands_AddBands()

```
#include <commctrl.h>
```

Adds bands to the CommandBands control.

Example:

```
REBARBANDINFO rbiInfo[2];
UNIT iBandCount = 2; //add 2 bands
rbiInfo[0].cbSize = sizeof (REBARBANDINFO);
rbiInfo[0].fMask = RBBIM_ID | RBBIM_STYLE;
```

```
rbiInfo[0].fStyle = RBBS_GRIPPERALWAYS | RBBS_FIXEDSIZE;
rbiInfo[0].wID = 0;
rbiInfo[0].iImage = 0;
rbiInfo[1].cbSize = sizeof (REBARBANDINFO);
rbiInfo[1].fMask = RBBIM_ID | RBBIM_STYLE;
rbiInfo[1].fStyle = RBBS_GRIPPERALWAYS | RBBS_FIXEDSIZE;
rbiInfo[1].wID = 1;
rbiInfo[1].iImage = 1;
CommandBands_AddBands(hBands, hInstance, iBandCount, &rbiInfo);
```

Where:

- `hBands` is a handle to the CommandBands control.
- `hInstance` is a handle to the application instance.
- `iBandCount` is an integer specifying the number of bands to add.
- `rbiInfo` is an array containing `iBandCount` `REBARBANDINFO` structures.

CommandBands_Create()

```
#include <commctrl.h>
```

Creates a new CommandBands control.

Example:

```
    DWORD dwStyles = CCS_VERT | RBS_VERTICALGRIPPER; //create a
vertical band with the proper vertical gripper
    WORD wID = ID_CMDBAND;
    hBands = CommandBands_Create (hInstance, hWnd, wID, dwStyles,
hImgList);
```

Where:

- `hBands` is the handle of the newly created CommandBands control if the function is successful.
- `hWnd` is the handle to the window that will own the CommandBands control.
- `hInstance` is a handle to the application instance.
- `wID` is a `WORD` specifying the numeric ID of the CommandBands control to be used in notification messages for that CommandBands control.

- `dwStyles` is a `DWORD` specifying how the CommandBands control is to appear. This value a combination of the following constants:

  CCS_VERT CommandBands control should appear vertically.

  RBS_AUTOSIZE CommandBands control's size will change automatically as needed.

  RBS_BANDBORDERS CommandBands control will be drawn with borders.

  RBS_FIXEDORDER The order of the CommandBands control's individual bands cannot be changed.

  RBS_SMARTLABELS CommandBands control's bands will only show their icon when minimized.

  RBS_VARHEIGHT CommandBands control will occupy the minimum possible vertical space.

  RBS_VERTICALGRIPPER CommandBands control will have a vertical gripper; for vertical CommandBands only.

- `hImgList` is a handle to an Image List.

CommandBands_GetCommandBar()

```
#include <commctrl.h>
```

Retrieves the handle to a CommandBar child control in a band of the CommandBands control.

Example:

```
HWND CommandBands_GetCommandBar( HWND hwndCmdBands, UINT uBand);
```

Where:

- `hwndCmdBands` is the handle to a CommandBand.
- `uBand` is a `UINT` zero-based index specifying which CommandBar within the CommandBand you want to retrieve the handle of.

WARNING The uBand index value applies to the currently displayed order of the CommandBars and may be altered by user repositioning.

CommandBands_GetRestoreInformation()

```
#include <commctrl.h>
```

Retrieves information to restore a band's current state.

Example:

```
CommandBands_GetRestoreInformation(hBands, uBand, &cbrRestoreInfo);
```

Where:

- hBands is a handle to the CommandBands' control window.
- uBands is the index of the CommandBands control whose restore info the function should retrieve.
- cbrRestoreInfo is a COMMANDBANDSRESTOREINFO structure that the function will use to return the information.

The LPCOMMANDBANDSRESTOREINFO is defined as follows:

```
typedef struct tagCOMMANDBANDSRESTOREINFO {
UINT cbSize;
UINT wID;
UINT fStyle;
UINT cxRestored;
BOOL fMaximized;
} COMMANDBANDSRESTOREINFO;
```

CommandBands_Height()

```
#include <commctrl.h>
```

Returns the height (in pixels) of a CommandBands control.

Example:

```
CommandBands_Height(hBands);
```

Where:

- hBands is a handle to the CommandBands control you want to get the height of.

CommandBands_IsVisible()

```
#include <commctrl.h>
```

Determines whether or not the CommandBands control is visible.

Example:

```
BOOL CommandBands_IsVisible (hBands);
```

Where:

- hBands is a handle to the CommandBands control.

CommandBands_Show()

```
#include <commctrl.h>
```

Shows or hides the CommandBands control.

Example:

```
bShow = TRUE;
CommandBands_Show(hBands, bShow);
```

Where:

- hBands is a handle to the CommandBands control.
- bShow is a BOOL value indicating whether the function should show (TRUE) or hide (FALSE) the CommandBands control.

CommandBar_AddAdornments()

```
#include <commctrl.h>
```

Adds the Close button (X) to the CommandBar. You can also use it to add the Help button (?) and the OK button.

Example:

```
DWORD dwFlags = CMDBAR_HELP | CMDBAR_OK;
CommandBar_AddAdornments(hCB, dwFlags, 0);
```

Where:

- hCB is the handle to the CommandBar.

- dwFlags is a DWORD specifying one or more of the following values:

  CMDBAR_HELP Add the Help (?) button.

  CMDBAR_OK Add the OK button.

NOTE The last parameter of this function is reserved and should always be 0.

CommandBar_AddBitmap()

```
#include <commctrl.h>
```

Adds a bitmap containing button images to the available CommandBar button images.

Example:

```
hCB = CommandBar_Create(hInstance, hwnd, 1);
iBmpID = IDB_BUTTONS;
iNumImages = 3;
CommandBar_AddBitmap(hCB, hInstance, iBmpID, iNumImages, 0, 0);
```

Where:

- hCB is a handle to the CommandBar.
- hInstance is an instance handle to the module containing the bitmap.
- iBmpID is the identifier of the bitmap resource.
- iNumImages is the number of 16 × 16 images contained in the bitmap.

NOTE The last two parameters of this function are reserved; it appears that it is safe to pass any integer values.

CommandBar_AddButtons()

```
#include <commctrl.h>
```

Adds buttons to an existing CommandBar.

Example:

```
CommandBar_AddButtons(hCB, iNumButtons, tbButtons);
```

Where:

- hCB is a handle to the CommandBar.
- iNumButtons is the number of buttons to be added.
- tbButtons is an array of TBBUTTON structures.

The TBBUTTON structure is defined as follows:

```
typedef struct _TBBUTTON {
int iBitmap;
int idCommand;
BYTE fsState;
BYTE fsStyle;
DWORD dwData;
int iString;
} TBBUTTON;
```

CommandBar_AddToolTips()

```
#include <commctrl.h>
```

Adds ToolTips hints to the buttons of a CommandBar.

Example:

```
CommandBar_AddToolTips(hCB, uNumTips, lpszTips);
```

Where:

- hCB is a handle to the CommandBar.
- uNumTips is the number of ToolTip strings to add.
- lpszTips is an array containing uNumTips strings containing the tips.

CommandBar_Create()

```
#include <commctrl.h>
```

Creates a CommandBar control.

Example:

```
iCmdID = 1;
hCB = CommandBar_Create(hInstance, hWnd, iCmdID);
```

Where:

- hInstance is the instance handle of the application.
- hWnd is the handle of the parent window of the CommandBar to be created.
- iCmdID is an integer used to identify this CommandBar from other CommandBars used in the application.

CommandBar_DrawMenuBar()

```
#include <commctrl.h>
```

Redraws the CommandBar after one of its menus has been modified.

Example:

```
CommandBar_DrawMenuBar(hCB, iMenuBar);
```

Where:

- hCB is a handle to a CommandBar.
- iMenuBar is the index of the menu within the CommandBar that you wish to redraw.

CommandBar_GetMenu()

```
#include <commctrl.h>
```

Retrieves the handle to a menu of the CommandBar.

Example:

```
hCmdBarMenu = CommandBar_GetMenu(hCB,  iButton);
```

Where:

- hCB is a handle to a CommandBar.
- iButton is the index of the CommandBar separator button holding the menu; in most cases, this will be 0.

CommandBar_Height()

```
#include <commctrl.h>
```

Returns the height of the CommandBar in pixels.

Example:

```
CommandBar_Height(hCB);
```

Where:

- hCB is a handle to the CommandBar whose height you want to know.

CommandBar_InsertButton()

```
#include <commctrl.h>
```

Inserts a button into the CommandBar.

Example:

```
CommandBar_InsertButton(hCB, iButton, &tbButton);
```

Where:

- hCB is a handle to a CommandBar.
- iButton is the index of the button that this new button will be placed to the left of.
- tbButton is a TBBUTTON structure.

CommandBar_InsertComboBox()

```
#include <commctrl.h>
```

Inserts a combo box control into the CommandBar.

Example:

```
iComboID = 300;
iButton = 4;
hCombo = CommandBar_InsertComboBox(hCB, hInstance, 120,
CBS_DROPDOWNLIST, ID_CMDCOMBO, iButton);
```

Where:

- hCombo is the handle to the newly created combo box if the function is successful.
- hCB is a handle to a CommandBar.
- hInstance is the instance handle of the application.

- CBS_DROPDOWNLIST is a constant specifying the style of combo box to be created. Any valid "Window Style" constants can be combined for this parameter.
- iComboID is an integer used to identify this combo box from other combo boxes used in the application.
- iButton is the index of the button that the combo box will be placed to the left of.

CommandBar_InsertMenubar()

```
#include <commctrl.h>
```

Inserts a MenuBar into the CommandBar.

Example:

```
CommandBar_InsertMenubar(hCB,  hInstance, idMenu, iButton);
```

Where:

- hCB is a handle to a CommandBar.
- hInstance is the instance handle of the application.
- idMenu is an integer used to identify this combo box from other combo boxes used in the application.
- iButton is the index of the button that the combo box will be placed to the left of.

CommandBar_InsertMenubarEx()

```
#include <commctrl.h>
```

Inserts a menu into a CommandBar.

Example:

```
CommandBar_InsertMenubarEx(hCB, hInstance, lpszMenuName, iButton);
```

Where:

- hCB is the handle to the CommandBar.
- hInstance is the instance handle of the application.
- lpszMenuName is the name of the menu resource.

- `iButton` is the index of the button that the combo box will be placed to the left of.

TIP

The main difference between `CommandBar_InsertMenubar()` and `CommandBar_InsertMenubarEx()` is that with the "`-Ex()`" version, the menu to be inserted can be identified by a resource name or by its handle if it has already been created.

CommandBar_IsVisible()

```
#include <commctrl.h>
```

Determines whether or not the CommandBar is visible.

Example:

```
CommandBar_IsVisible(hCB);
```

Where:

- `hCB` is the handle to the CommandBar.

CommandBar_Show()

```
#include <commctrl.h>
```

Shows or hides the CommandBar.

Example:

```
bShow = TRUE;
BOOL CommandBar_Show(hCB, bShow);
```

Where:

- `hCB` is a handle to the CommandBar control.
- `bShow` is a `BOOL` value indicating whether the function should show (`TRUE`) or hide (`FALSE`) the CommandBar control.

CreateAddressBook()

```
#include <addrstor.h>
```

Creates the Contacts database if it doesn't exist.

Example:

```
int iSortOrderCount = 4;
HHPRTAG hPrTagList[4] = {HHPR_COMPANY_NAME, HHPR_FILEAS,
HHPR_GIVEN_NAME, HHPR_TITLE};
CreateAddressBook (hPrTagList, iSortOrderCount);
```

Where:

- `hPrTagList` is an array of `HHPRTAG` constants specifying the possible sort orders for the Contacts database. It may contain up to four `HHPRTAG`s.
- `iSortOrderCount` is an integer specifying the number of `HHPRTAG`s in the `hPrTagList` array.

CreateFileForMapping()

```
#include <winbase.h>
```

Creates a memory-mapped file.

Example:

```
TCHAR szFileName[MAX_PATH] = TEXT("\\data.txt");
wAccess = GENERIC_WRITE;
wShare = 0;
dwCreate = CREATE_ALWAYS;
dwFlags = FILE_ATTRIBUTE_NORMAL;
hSrc = CreateFileForMapping(
szFileName,
wAccess,
wShare,
NULL,
dwCreate,
dwFlags,
NULL);
```

Where:

- `szFileName` is a string specifying the name of the file to open.
- `wAccess` is a `WORD` specifying the type of access (`GENERIC_READ` and/or `GENERIC_WRITE`).
- `wShare` is a `WORD` specifying under what conditions another application may open or modify the file while you are using it (`0` for no access, `FILE_SHARE_READ` to permit read access, and `FILE_SHARE_WRITE` to permit write access).
- `dwCreate` is a `DWORD` specifying whether you are attempting to open an existing file or create a file (CREATE_ALWAYS always creates the file, OPEN_EXISTING attempts to open an existing file, etc.).
- `dwFlags` is a `DWORD` specifying additional file attributes, such as the archive, hidden, or read-only flags.

TIP The fourth parameter specifies security attributes and the seventh specifies a template file. Both are always NULL on CE.

DeleteAddressCard()

```
#include <addrstor.h>
```

Deletes the specified entry from the Contacts database.

Example:

```
DeleteAddressCard(oidToDelete);
```

Where:

- `oidToDelete` is a CEOID specifying the entry (record) to delete.

DeleteAndRenameFile()

```
#include <winbase.h>
```

Copies source file to destination file, then deletes source file.

Example:

```
TCHAR szSource[MAX_PATH] = TEXT("\\MyData.txt");
TCHAR szDest[MAX_PATH] = TEXT("\\Archived\\MyData.txt");
DeleteAndRenameFile(szDest, szSource);
```

Where:

- szDest is a string specifying the name of the destination file.
- szSource is a string specifying the name of the source file.

DeregisterDevice()

```
#include <winbase.h>
```

Deregisters a device previously registered with a call to RegisterDevice().

Example:

```
DeregisterDevice(hDev);
```

Where:

- hDev is a handle to a device.

EnableEUDC()

```
#include <wingdi.h>
```

Enables or disables end user–defined characters (EUDCs). Applies to Asian-language versions of Windows CE.

Example:

```
BOOL bEnable;
if (IDYES == MessageBox(hWnd, TEXT("Do you want enable EUDCs?"),
TEXT("Question"), MB_YESNO))
{
    bEnable = TRUE;
}
else
{
    bEnable = FALSE;
}
EnableEUDC(bEnable);
```

Where:

- `bEnable` is a `BOOL` specifying whether EUDCs are to be enabled (`TRUE`) or disabled (`FALSE`).

EnableHardwareKeyboard()

```
#include <winuser.h>
```

If the device has a hardware keyboard, this function can be used to turn it on or off.

Example:

```
BOOL bEnable;
if (IDYES == MessageBox(hWnd, TEXT("Do you want to turn on the
keyboard again?"), TEXT("Question"), MB_YESNO))
{
    bEnable = TRUE;
}
else
{
    bEnable = FALSE;
}
EnableHardwareKeyboard(bEnable);
```

Where:

- `bEnable` is a `BOOL` specifying whether EUDCs are to be enabled (`TRUE`) or disabled (`FALSE`).

TIP

This function might be useful in situations where the user is concerned about accidentally striking keys (for example, when they are entering data with the stylus for an extended period of time).

FindFirstEntry()

```
#include <addrstor.h>
```

Searches the object store for the first item that *follows* the specified value; returns the `oid` of the matching item.

Example:

```
if (OpenAddressBook (NULL, HHPR_FILEAS))
{
    TCHAR szToMatch[MAX_PATH] = TEXT("J"); //find first *following*
value
    dwFlags = 0;
    oid = FindFirstEntry(szToMatch, HHPR_FILEAS, &iIndex, dwFlags);
```

Where:

- `oid` is the CEOID of the first item following the searched-for value.
- `szToMatch` is a string specifying a value to search for.
- `HHPR_COMPANY_NAME` is an HHPRTAG, one of numerous possible constants specifying the property (field) to search on. Other possible values include
  - `HHPR_ANNIVERSARY`
  - `HHPR_ASSISTANT_NAME`
  - `HHPR_ASSISTANT_TELEPHONE_NUMBER`
  - `HHPR_BIRTHDAY`
  - `HHPR_BUSINESS_FAX_NUMBER`
  - `HHPR_CAR_TELEPHONE_NUMBER`
  - `HHPR_CATEGORY`
  - `HHPR_CHILDREN_NAME`
  - `HHPR_COMPANY_NAME`
  - `HHPR_CUSTOM_DISPLAY_FIELDS`
  - `HHPR_DEPARTMENT_NAME`
  - `HHPR_EMAIL1_EMAIL_ADDRESS`
  - `HHPR_EMAIL2_EMAIL_ADDRESS`
  - `HHPR_EMAIL3_EMAIL_ADDRESS`
  - `HHPR_GENERATION`
  - `HHPR_GIVEN_NAME`
  - `HHPR_HOME2_TELEPHONE_NUMBER`

- HHPR_HOME_ADDRESS_CITY
- HHPR_HOME_ADDRESS_COUNTRY
- HHPR_HOME_ADDRESS_POSTAL_CODE
- HHPR_HOME_ADDRESS_STATE
- HHPR_HOME_ADDRESS_STREET
- HHPR_HOME_FAX_NUMBER
- HHPR_HOME_TELEPHONE_NUMBER
- HHPR_MIDDLE_NAME
- HHPR_MOBILE_TELEPHONE_NUMBER
- HHPR_NAME_PREFIX
- HHPR_NOTES
- HHPR_OFFICE_ADDRESS_CITY
- HHPR_OFFICE_ADDRESS_COUNTRY
- HHPR_OFFICE_ADDRESS_POSTAL_CODE
- HHPR_OFFICE_ADDRESS_STATE
- HHPR_OFFICE_ADDRESS_STREET
- HHPR_OFFICE_LOCATION
- HHPR_OFFICE_TELEPHONE_NUMBER
- HHPR_OFFICE2_TELEPHONE_NUMBER
- HHPR_OTHER_ADDRESS_CITY
- HHPR_OTHER_ADDRESS_COUNTRY
- HHPR_OTHER_ADDRESS_POSTAL_CODE
- HHPR_OTHER_ADDRESS_STATE
- HHPR_OTHER_ADDRESS_STREET
- HHPR_PAGER_NUMBER
- HHPR_SPOUSE_NAME

  - HHPR_SURNAME
  - HHPR_TITLE
  - HHPR_WEB_PAGE
  - HHPR_YOMI_NAME //Japanese CE only
  - HHPR_YOMI_COMPANY //Japanese CE only
- iIndex is an integer used by the function to return the index of the matching item.
- dwFlags is either 0 or FFE_CONTAINS. Specifying FFE_CONTAINS searches for the first value following and containing szToMatch.

FreeAddressCard()

```
#include <addrstor.h>
```

Frees memory allocated by calls to OpenAddressCard() or GetAddressCardProperties().

Example:

```
AddressCard ac;
GetAddressCardProperties(oidCard, &ac, 8, rghhProp);
//...do something...
FreeAddressCard(&ac);
```

Where:

- ac is an AddressCard structure.

The AddressCard structure is defined as follows:

```
typedef struct _AddressCard {
SYSTEMTIME stBirthday;
SYSTEMTIME stAnniversary;
TCHAR *pszBusinessFax;
TCHAR *pszCompany;
TCHAR *pszDepartment;
TCHAR *pszEmail;
TCHAR *pszMobilePhone;
TCHAR *pszOfficeLocation;
TCHAR *pszPager;
TCHAR *pszWorkPhone;
```

```
TCHAR *pszTitle;
TCHAR *pszHomePhone;
TCHAR *pszEmail2;
TCHAR *pszSpouse;
TCHAR *pszNotes;
TCHAR *pszEmail3;
TCHAR *pszHomePhone2;
TCHAR *pszHomeFax;
TCHAR *pszCarPhone;
TCHAR *pszAssistant;
TCHAR *pszAssistantPhone;
TCHAR *pszChildren;
TCHAR *pszCategory;
TCHAR *pszWebPage;
TCHAR *pszWorkPhone2;
TCHAR *pszNamePrefix;
TCHAR *pszGivenName;
TCHAR *pszMiddleName;
TCHAR *pszSurname;
TCHAR *pszGeneration;
TCHAR *pszHomeAddrStreet;
TCHAR *pszHomeAddrCity;
TCHAR *pszHomeAddrState;
TCHAR *pszHomeAddrPostalCode;
TCHAR *pszHomeAddrCountry;
TCHAR *pszOtherAddrStreet;
TCHAR *pszOtherAddrCity;
TCHAR *pszOtherAddrState;
TCHAR *pszOtherAddrPostalCode;
TCHAR *pszOtherAddrCountry;
TCHAR *pszOfficeAddrStreet;
TCHAR *pszOfficeAddrCity;
TCHAR *pszOfficeAddrState;
TCHAR *pszOfficeAddrPostalCode;
TCHAR *pszOfficeAddrCountry;
BYTE *rgbReserved[84]
} AddressCard;
```

FtpCommand()

```
#include <wininet.h>
```

Sends the specified command to an FTP server.

Example:

```
BOOL bResponse = TRUE;
char szCmd[MAX_PATH] = "HELP\r\n";
DWORD dwFlags = FTP_TRANSFER_TYPE_ASCII;
DWORD dwContext = 100;
WINAPI FtpCommand(hFtpSession, bResponse, dwFlags, szCmd,
dwContext);
```

Where:

- `hFtpSession` is a handle to an open FTP session.
- `bResponse` is a `BOOL` indicating whether the command being sent will generate a response; in this example, it will generate a response.
- `dwFlag` is a `DWORD` specifying the type of data being sent. Can be one (but not both) of the following:

  FTP_TRANSFER_TYPE_ASCII ASCII data transfer

  FTP_TRANSFER_TYPE_BINARY Binary data transfer
- `szCmd` is a string containing the command to send to the server.
- `dwContext` is a `DWORD` used to uniquely identify this command for use with the status callback functions.

GetAddressCardIndex()

```
#include <addrstor.h>
```

Retrieves index of an address card in the contacts database, given its `CEOID`.

Example:

```
iIndex = GetAddressCardIndex(oidAC);
```

Where:

- `iIndex` is an integer that will receive the index of the address card or a value indicating an error. This zero-based index is based on the current sort order of the Address Book.
- `oidAC` is the `CEOID` of the address card whose index the function will retrieve.

GetAddressCardOid()

```
#include <addrstor.h>
```

Retrieves the CEOID of an address card given its index.

Example:

```
oidAC = GetAddressCardOid(iIndex);
```

Where:

- oidAC is the CEOID of the address card whose index is being passed to the function.
- iIndex is an integer specifying the index of the address card.

GetAddressCardProperties()

```
#include <addrstor.h>
```

Opens an address card and returns its properties.

Example:

```
AddressCard ac;
int iPropCount = 7; //retrieve 7 properties
HHPRTAG ptProps[7] = {HHPR_SURNAME, HHPR_GIVEN_NAME, HHPR_TITLE,
HHPR_OFFICE_LOCATION, HHPR_OFFICE_TELEPHONE_NUMBER,
HHPR_BUSINESS_FAX_NUMBER, HHPR_ANNIVERSARY};
GetAddressCardProperties(oidCard,&ac,iPropCount, ptProps);
```

Where:

- oidCard is the CEOID of the card whose properties the function will retrieve.
- ac is an AddressCard structure. The members of the AddressCard structure that correspond to the properties requested will contain the values of those properties when the function returns.
- iPropCount is an integer specifying the number of properties requested.
- ptProps is an array containing iPropCount HHPRTAG values. See FindFirstEntry() for a list of possible values for this parameter.

GetClipboardDataAlloc()

```
#include <winuser.h>
```

Allocates memory for and returns a handle to the data in the clipboard. This handle to memory must be freed later by the application via a call to LocalFree().

Example:

```
UINT uTypeOfData = CF_TEXT;
TCHAR *szClipData;
szClipData = (TCHAR *) GetClipboardDataAlloc(uTypeOfData);
```

Where:

- szClipData is a buffer (in this case, a TCHAR buffer) that will point to the data in the clipboard when the function returns.
- uTypeOfData is UINT specifying the format of the clipboard data. Common values include

  CF_UNICODETEXT The clipboard contains (Unicode) text data.

  CF_BITMAP The clipboard contains bitmap data.

GetColumnProperties()

```
#include <addrstor.h>
```

Retrieves an array of properties representing the properties on which the Contacts database can be sorted.

Example:

```
    int iColCount = 4; //maximum # of properties the function can
return.
    HHPRTAG ptProps[4]; //reserve enough space for maximum # of
properties
    GetColumnProperties(&ptProps[0], &iColCount);
```

Where:

- ptProps is an array of HHPRTAG property specifiers. See GetAddressCardProperties() for a list of possible HHPRTAG values. The function returns a maximum of four HHPRTAG values in the array.
- iColCount is an integer specifying the number of HHPRTAG props actually returned by the function.

GetMatchingEntry()

```
#include <addrstor.h>
```

Searches specified text-based property of the Contacts database for a matching value.

Example:

```
CEOID oidMatch;
HHPRTAG ptProp;
TCHAR szToMatch[MAX_PATH] = TEXT("billg@microsoft.com"); //find
matching value
iToMatchLen = _tcslen(szToMatch);
TCHAR szPropToSearch[MAX_PATH] = TEXT("Internet");
if (OpenAddressBook (NULL,NULL))
{
    GetMatchingEntry(szToMatch, iToMatchLen, szPropToSearch,
&oidMatch, &ptProp);
    //...
```

Where:

- `szToMatch` is a string specifying a value to search for.
- `iToMatchLen` is an integer specifying the length of `szToMatch`.
- `szPropToSearch` is the name of the Contacts database property (field) to search.
- `oidMatch` is a `CEOID` the function uses to return the `CEOID` of the matching record.
- `ptProp` is an `HHPRTAG` value the function uses to return the `HHPRTAG` of the matching record. See `GetAddressCardProperties()` for a list of possible `HHPRTAG` values.

GetMessageSource()

```
#include <winuser.h>
```

Used to determine the origin of a keyboard-related message.

Example:

```
UINT iMsgSource;
iMessageSource = GetMessageSource();
```

Where:

- `iMessageSource` is a `UINT` value, which when the function returns will contain one of the following values:

  MSGSRC_SOFTWARE_POST Another application generated the sent or posted the keyboard message to our application.

  MSGSRC_HARDWARE_KEYBOARD The user actually struck a key on the keyboard to generate this message.

  MSGSRC_UNKNOWN Source of the keyboard message is unknown.

GetMouseMovePoints()

```
#include <winuser.h>
```

Retrieves an array of `POINT`s indicating all of the points the mouse or stylus has been at for the current mouse-/stylus-related message. The advantage of this is that typically these points are recorded at too high a rate for an application to properly handle them. This function, then, allows for higher-resolution tracking of the stylus or mouse.

Example:

```
    POINT ptPointArray[10]; //reserve room for up to 10 POINT
structures
    UINT uPointCount = 10; //# of elements in array
    UINT uPointsRetrieved;
    GetMouseMovePoints(&ptPointArray, uPointCount, &uPointsRetreived);
```

Where:

- `ptPointArray` is an array of `POINT` structures.
- `uPointsCount` is a `UINT` specifying the number of `POINT` structures in `ptPointArray`.
- `uPointsRetrieved` is a `UINT` the function will use to indicate how many points it actually wrote to `ptPointArray`.

WARNING This function does not exist under the CE emulators!

GetNumberOfAddressCards()

```
#include <addrstor.h>
```

Retrieves the number of address cards (records) in the Contacts database.

Example:

```
int iAddressCardCount;
iAddressCardCount = GetNumberOfAddressCards();
```

Where:

- `iAddressCardCount` is an integer that will receive the return value indicating the number of address cards.

GetPropertyDataStruct()

```
#include <addrstor.h>
```

Retrieves information about a property in the Contacts database.

Example:

```
int fFlag = GPDS_PROPERTY;
ULONG uPropID = HHPR_COMPANY_NAME;
PropertyDataStruct pds;
GetPropertyDataStruct(fFlag, uPropID, &pds);
```

Where:

- `fFlag` is an integer specifying how the function should search for the property. Possible values are

  GPDS_INDEX Treat the function's second parameter as the index of the property to search for.

  GPDS_NAME Treat the function's second parameter as a string specifying the name of the function.

  GPDS_PROPERTY Treat the function's second parameter as an HHPROPTAG value, as in the example above.

- `uPropID` is a `ULONG` value. Its purpose depends on the value of the first parameter.
- `pds` is a `PropertyDataStruct` structure the function will use to return information about the specified property.

GetSortOrder()

```
#include <addrstor.h>
```

Retrieves the property on which the Contacts database is currently sorted. Returns the property tag of this property.

Example:

```
BOOL bTrueSort = TRUE;
HHPRTAG hptPropSortOrder;
hptPropSortOrder = GetSortOrder(bTrueSort);
```

Where:

- `hptPropSortOrder` is an `HHPRTAG` value that is being used to store the return value of the function, identifying the property on which the Contacts database is currently sorted.
- `bTrueSort` is a `BOOL` specifying whether the function should return the property tag of the property the database is actually sorted on (`TRUE`), or the property tag of the property the user *thinks* the database is sorted on (`FALSE`). For example, although the database might be sorted on the `HHPR_GIVEN_NAME`, it would appear to the user to be sorted on the `HHPR_FIRST_LAST_NAME` property.

GetStoreInformation()

```
#include <winbase.h>
```

Returns information about free space of the CE device's object store.

Example:

```
STORE_INFORMATION siInfo;
CeGetStoreInformation(&siInfo);
```

Where:

- `siInfo` is a `STORE_INFORMATION` structure to be filled with the information about the object store.

The `STORE_INFORMATION` structure is defined as follows:

```
typedef struct STORE_INFORMATION {
DWORD dwStoreSize;
DWORD dwFreeSize;
} STORE_INFORMATION
```

GetSystemPowerStatusEx()

```
#include <winbase.h>
```

Retrieves information about the CE device's power source.

Example:

```
bMostCurrentInfo = FALSE;
CeGetSystemPowerStatusEx(&psPowerStatus, bMostCurrentInfo);
```

Where:

- psPowerStatus is a SYSTEM_POWER_STATUS_EX structure.
- bMostCurrentInfo is a BOOL value indicating whether the function should query the device driver directly for the most up-to-date information, or whether it should use cached information that may be several seconds old.

The SYSTEM_POWER_STATUS_EX structure is defined as follows:

```
typedef struct _SYSTEM_POWER_STATUS_EX {
BYTE ACLineStatus;
BYTE BatteryFlag;
BYTE BatteryLifePercent;
BYTE Reserved1;
DWORD BatteryLifeTime;
DWORD BatteryFullLifeTime;
BYTE Reserved2;
BYTE BackupBatteryFlag;
BYTE BackupBatteryLifePercent;
BYTE Reserved3;
DWORD BackupBatteryLifeTime;
DWORD BackupBatteryFullLifeTime;
} SYSTEM_POWER_STATUS_EX;
```

ImageList_Duplicate()

```
#include <commctrl.h>
```

Creates a duplicate or copy of an existing image list.

Example:

```
HIMAGELIST hImgListNew;
hImgListNew = ImageList_Duplicate(hImgListOld);
```

Where:

- `hImgListNew` is a handle to an image list. If the function is successful, `hImgListNew` will be a handle to a new image list containing the same images as `hImgListOld`.
- `hImgListOld` is a handle to the image list to be copied.

InitCommonControlsEx()

```
#include <commctrl.h>
```

Initializes the `commctrl.dll` library and makes the requested controls available to the calling application.

Example:

```
INITCOMMONCONTROLSEX icInitCommCtrl;
icInitCommCtrl.dwSize = sizeof(INITCOMMONCONTROLSEX);
icInitCommCtrl.dwICC = ICC_TREEVIEW_CLASSES;
InitCommonControlsEx(&icInitCommCtrl);
```

Where:

- `icInitCommCtrl` is an `INITCOMMONCONTROLSEX` structure. The `dwICC` member of this structure specifies which controls are being requested from the `commctrl.dll`. The `dwICC` member can be any combination of the following values:

  ICC_BAR_CLASSES "Bar" controls: toolbar, status bar, trackbar, and CommandBar

  ICC_COOL_CLASSES Rebar control

  ICC_DATE_CLASSES Date and date-time picker controls

  ICC_LISTVIEW_CLASSES Listview and header controls

  ICC_PROGRESS_CLASS ProgressBar control.

  ICC_TAB_CLASSES Tab controls

  ICC_TREEVIEW_CLASSES Treeview control

  ICC_UPDOWN_CLASS Up-down control

The INITCOMMONCONTROLSEX structure is defined as follows:

```
typedef struct tagINITCOMMONCONTROLSEX {
DWORD dwSize;
DWORD dwICC;
} INITCOMMONCONTROLSEX;
```

InitInkX()

```
#include <richink.h>
```

Initializes the Ink control libraries; must be called before attempting to use the Ink control.

Example:

```
InitInkX();
```

NOTE This function takes no parameters and does not return a value.

WARNING You must always call `InitCommonControls()` before calling `InitInkX()`.

InitHTMLControl()

```
#include <htmlctrl.h>
```

Initializes the HTML control.

Example:

```
InitHTMLControl(hInstance);
```

Where:

- `hInstance` is a handle to the application's instance.

TIP Although the documentation says, "An application must load the HTML control library `Htmlview.dll`" before calling `InitHTMLControl()`, this is not necessary. Simply calling `InitHTMLControl()` is enough.

InterlockedTestExchange()

```
#include <winbase.h>
```

Conditionally sets the value of a LONG variable, guaranteeing that the values will be successfully exchanged via synchronization in the case of multiple threads accessing the variables.

Example:

```
LONG lTarget = 444;
LONG lOldVal = 444;
LONG lNewVal = 42;
InterlockedTestExchange(lTarget, lOldVal, lNewVal);
```

Where:

- `lTarget` is a LONG variable whose value the function will conditionally replace.
- `lOldVal` is a LONG value the function uses to determine whether or not it should replace the value of `lTarget`. If `lTarget` and `lOldVal` are equal, then the function will replace the value of `lTarget`.
- `lNewVal` is a LONG value the function will assign to `lTarget` providing that `lTarget` is equal to `lOldVal`.

MailClose()

```
#include <msgstore.h>
```

Closes an open message store handle.

Example:

```
MailClose(hMsgStore);
```

Where:

- `hMsgStore` is an open handle to the message store.

MailDelete()

```
#include <msgstore.h>
```

Deletes a mail message from the message store.

Example:

```
MailDelete(hMsgStore, &mmMsg);
```

Where:

- `hMsgStore` is an open handle to the message store.
- `mmMsg` is a `MailMsg` structure specifying the message to be deleted.

The `MailMsg` structure is defined as follows:

```
typedef struct MailMsg {
DWORD dwMsgId;
DWORD dwFlags;
DWORD dwMsgLen;
WORD wBodyLen;
FILETIME ftDate;
LPWSTR szSvcId;
LPWSTR szSvcNam;
WCHAR *pwcHeaders;
LPWSTR szBody;
CEOID oid;
HANDLE hHeap
} MailMsg;
```

MailDeleteAttachment()

```
#include <msgstore.h>
```

Deletes a mail message's attachment.

Example:

```
BOOL bDeleteEntire = TRUE;
MailDeleteAttachment (hMsgStore, &mmMsg, maAttach, bDeleteEntire);
```

Where:

- `hMsgStore` is an open handle to the message store.
- `mmMsg` is a `MailMsg` structure specifying the mail message to which the attachment belongs.
- `maAttach` is a `MailAtt` structure specifying the attachment to be deleted.
- `bDeleteEntire` specifies whether the entire attachment is to be deleted (`TRUE`) or whether the header of the attachment should be retained (`FALSE`).

The MailAtt structure is defined as follows:

```
typedef struct MailAtt_s {
UINT uiAttachmentNumber;
DWORD dwFlags;
ULONG ulCharacterPosition;
ULONG ulSize;
LPWSTR szOriginalName;
LPWSTR szLocalName;
} MailAtt;
```

MailError()

```
#include <msgstore.h>
```

Comparable to GetLastError(); returns a numeric code for the last error that was caused by a Message Store function.

Example:

```
int iError;
iError = MailError(hMsgStore);
```

Where:

- iError is an integer used to store the result of the function.
- hMsgStore is an open handle to the message store.

MailErrorMsg()

```
#include <msgstore.h>
```

Comparable to FormatMessage(); formats a string error message based on the last error that was caused by a Message Store function. Also provides the numeric identifier of the last error as its return value.

Example:

```
int iError;
int iLineNo;
TCHAR szError[MAX_PATH];
iError = MailError(hMsgStore, szError, MAX_PATH, &iLineNo);
```

Where:

- iError is an integer used to store the result of the function.
- hMsgStore is an open handle to the message store.

- szError is a string that, if the function is sucessful, will contain a text message indicating the error that occurred.
- MAX_PATH specifies the length, in bytes, of szError.
- iLineNo is an integer the function uses to return the line number on which the error occurred.

WARNING This function is provided as an alternative to MailError().

MailFirst()

```
#include <msgstore.h>
```

Retrieves the first message matching the specified criteria.

Example:

```
if (MailOpen(&hMsgStore, FALSE))
{
    mmMsg.dwFlags = MAIL_FOLDER_INBOX | MAIL_FULL;
    MailFirst(hMsgStore, &mmMsg);
    //...
}
MailClose(hMsgStore);
```

Where:

- hMsgStore is an open handle to the message store.
- mmMsg is a MailMsg structure specifying the mail message to retrieve.

The MailMsg structure is defined as follows:

```
typedef struct MailMsg {
DWORD dwMsgId;
DWORD dwFlags;
DWORD dwMsgLen;
WORD wBodyLen;
FILETIME ftDate;
LPWSTR szSvcId;
LPWSTR szSvcNam;
WCHAR *pwcHeaders;
LPWSTR szBody;
```

```
CEOID oid;
HANDLE hHeap
) MailMsg;
```

MailFree()

```
#include <msgstore.h>
```

Frees memory allocated by calls to `MailFirst()`, `MailGet()`, or `MailNext()`.

Example:

```
if (MailOpen(&hMsgStore, FALSE))
{
    mmMsg.dwFlags = MAIL_FOLDER_INBOX | MAIL_FULL;
    MailFirst(hMsgStore, &mmMsg);
    //...
    //...done w/ message, so free memory
    MailFree(&mmMsg);
}
MailClose(hMsgStore);
```

Where:

- `hMsgStore` is an open handle to the message store.
- `mmMsg` is a `MailMsg` structure specifying the mail message to retrieve.

MailGet()

```
#include <msgstore.h>
```

Retrieves specified mail entry from the message store.

Example:

```
if (MailOpen(&hMsgStore, FALSE))
{
    memset(&mmMsg, 0, sizeof(MailMsg));
    mmMsg.dwFlags = MAIL_FOLDER_INBOX | MAIL_FULL;
    MailGet(hMsgStore, &mmMsg);
    //...
    //...done w/ message, so free memory
    MailFree(&mmMsg);
}
MailClose(hMsgStore);
```

Where:

- hMsgStore is an open handle to the message store.
- mmMsg is a MailMsg structure specifying the mail message to retrieve.

MailGetAttachment()

```
#include <msgstore.h>
```

Retrieves an attachment to a mail message.

Example:

```
MailAtt maAtt;
    MailMsg mmMsg;
    if (MailFirst(hMsgStore, &mmMsg))
    {
        // see if any attachments in this message
        if (mmMsg.dwFlags & MAIL_STATUS_ATTACHMENTS)
        {
            MailGetAttachment (hMsgStore, &mmMsg, &maAtt);
        }
    }
```

Where:

- hMsgStore is an open handle to the message store.
- mmMsg is a MailMsg structure specifying the mail message whose attachment you want to retrieve.
- maAttachment is a MailAtt structure the function will use to return the attachment.

The MailAtt structure is defined as follows:

```
typedef struct MailAtt_s {
UINT uiAttachmentNumber;
DWORD dwFlags;
ULONG ulCharacterPosition;
ULONG ulSize;
LPWSTR szOriginalName;
LPWSTR szLocalName;
} MailAtt;
```

MailGetField()

```
#include <msgstore.h>
```

Retrieves a specified header field from the mail message.

Example:

```
LPWSTR szFieldName = TEXT("Subject");
bGetName = FALSE;
MailGetField(&mmMsg, szFieldName, bGetName);
```

Where:

- mmMsg is a MailMessage previously retrieved with a call to MailFirst(), MailGet(), etc.
- szFieldName is the name of the mail message field to retrieve. Possible values include

  "Subject" Retrieves the subject field of the message.

  "To" Retrieves the recipient field of the message.

  "From" Retrieves the sender field of the message.

  "CC" Retrieves the carbon-copy-recipient field.

- bGetName is a BOOL indicating whether the function should retrieve the name of the field (e.g., "Subject") or the actual contents of the field. A TRUE value retrieves the field name; a FALSE value retrieves the contents.

MailGetFolderId()

```
#include <msgstore.h>
```

Retrieves the ID number of a mail folder from the message store.

Example:

```
BYTE iFolderID; //will receive ID of folder if successful
LPWSTR szFolderName = TEXT("Inbox");
MailGetFolderId(hMsgStore, &iFolderID, szFolderName);
```

Where:

- hMsgStore is an open handle to the message store.

- `iFolderID` is an integer the function will use to return the ID number of the folder specified.
- `szFolderName` is a string specifying the name of the folder whose ID you want to retrieve.

MailGetFolderName()

```
#include <msgstore.h>
```

Retrieves the name of a mail folder from the message store.

Example:

```
WCHAR szFolderName[MAX_PATH];
    int iLen = MAX_PATH;
    int iFolderID = MAIL_FOLDER_OUTBOX;
    MailGetFolderName(hMsgStore, &iFolderID, &iLen, szFolderName);
```

Where:

- `hMsgStore` is an open handle to the message store.
- `iFolderID` is an integer the function will use to return the ID number of the folder specified. Predefined system folder IDs include

  MAIL_FOLDER_INBOX Inbox Folder

  MAIL_FOLDER_OUTBOX Outbox Folder

  MAIL_FOLDER_SENT Sent Folder
- `iLen` is the maximum length of `szFolderName`.
- `szFolderName` is a string specifying the name of the folder whose ID you want to retrieve.

MailGetSort()

```
#include <msgstore.h>
```

Retrieves the information about the current sort order of the open message store.

Example:

```
MAILSORTINFO msiSortInfo;
if (MailGetSort (hMsgStore, &msiSortInfo))
{
    if (msiSortInfo == MAIL_SORT_FROM)
    {
        //do something...
    }
}
```

Where:

- `hMsgStore` is an open handle to the message store.
- `msiSortInfo` is a `MAILSORTINFO` structure.

The `MAILSORTINFO` structure is defined as follows:

```
typedef struct _MAILSORTINFO {
MAILSORTFIELD iSort;
BOOL fAscending;
int cMsgs;
} MAILSORTINFO;
```

MailGetSvcId()

```
#include <msgstore.h>
```

Retrieves the `CEOID` (object identifier) of a mail message from the message store based on the value of the `szSvcId` and `szSvcNam` (service identifier) fields of the `MailMsg` structure.

Example:

```
MailGetSvcId(hMsgStore, &mmMsg);
```

Where:

- `hMsgStore` is an open handle to the message store.
- `mmMsg` is a `MailMessage` structure that contains values for the `szSvcId` and `szSvcNam` fields.

TIP The "service identifier" is a string that uniquely identifies a mail message.

MailHeaderLen()

```
#include <msgstore.h>
```

Returns the size, in characters, of a mail message's header.

Example:

```
DWORD dwHeaderLen;
dwHeaderLen = MailHeaderLen(&mmMailMsg);
```

Where:

- `dwHeaderLen` is a `DWORD` used to store the return value of the function.
- `mmMailMsg` is a `MailMsg` structure.

MailLocalAttachmentLen()

```
#include <msgstore.h>
```

Returns the total size of a mail message's attachments.

Example:

```
DWORD dwTotalAttSize;
MailLocalAttachmentLen(hMsgStore, &mmMsg);
```

Where:

- `dwTotalAttSize` is a `DWORD` used to store the return value of the function.
- `mmMailMsg` is a `MailMsg` structure.

MailNext()

```
#include <msgstore.h>
```

Used in conjunction with `MailFirst()`; retrieves the next mail message matching the criteria specified in `MailFirst()`.

Example:

```
MailNext(hMsgStore, &mmMailMsg);
```

Where:

- `hMsgStore` is an open handle to the message store.
- `mmMsg` is a `MailMessage` structure that the function will use to return the next matching message.

MailOpen()

```
#include <msgstore.h>
```

Opens the message store and obtains a handle to the mail context.

Example:

```
HANDLE hMsgStore;
BOOL bAllowCreate = TRUE;
MailOpen(&hMsgStore, bAllowCreate);
```

Where:

- `hMsgStore` is a handle; if the function is successful, it will be a handle to the message store.
- `bAllowCreate` is a `BOOL` specifying whether you'd like the message store to be created if it does not exist.

MailOpenNotify()

```
#include <msgstore.h>
```

Opens the message store, obtains a handle to the mail context, and specifies that the calling application should receive notification if the message store is modified while the application is accessing it.

Example:

```
HANDLE hMsgStore;
BOOL bAllowCreate = TRUE;
MailOpenNotify(&hMsgStore, bAllowCreate, hWnd);
```

Where:

- `hMsgStore` is a handle; if the function is successful, it will be a handle to the message store.
- `bAllowCreate` is a `BOOL` specifying whether you'd like the message store to be created if it does not exist.
- `hWnd` is the handle to the window that should receive the notification messages.

MailPut()

```
#include <msgstore.h>
```

Retrieves specified mail entry from the message store.

Example:

```
if (MailOpen(&hMsgStore, FALSE))
{
    //...
    //fill in mmMsg structure here
    //...
    MailPut(hMsgStore, &mmMsg);
    //...
}
MailClose(hMsgStore);
```

Where:

- `hMsgStore` is an open handle to the message store.
- `mmMsg` is a `MailMsg` structure specifying the mail message to retrieve.

MailPutAttachment()

```
#include <msgstore.h>
```

Appends or modifies an attachment to a mail message.

Example:

```
MailAtt maAtt;
MailMsg mmMsg;
if (MailFirst(hMsgStore, &mmMsg))
{
    //...
    //make any modifications to maAtt here
    //
    MailPutAttachment (hMsgStore, &mmMsg, &maAtt);
}
```

Where:

- `hMsgStore` is an open handle to the message store.

- mmMsg is a MailMsg structure specifying the mail message whose attachment you want to retrieve.
- maAttachment is a MailAtt structure the function will use to return the attachment

NOTE To append the attachment to the mail message, set uiAttachmentNumber member of the MailAtt structure to be one higher than the uiAttachmentNumber of the last attachment of the message; to update an existing attachment, set uiAttachmentNumber to the uiAttachmentNumber value of the existing attachment.

MailPutFolder()

```
#include <msgstore.h>
```

Creates, deletes, or renames mail folders.

Example:

```
//create a folder
LPWSTR szFolderName = TEXT("Spam Mail");
BYTE iFolderID = 224;
MailPutFolder(hMsgStore, &iFolderID, szFolderName);
```

Where:

- hMsgStore is an open handle to the message store.
- iFolderID is an integer used to uniquely identify the folder; must be between 0 and 255.
- szFolderName is a string specifying the name of the folder to create or rename. To delete a folder, set this value to NULL.

MailRequestAttachment()

```
#include <msgstore.h>
```

Requests and/or downloads an attachment to a mail message that has not yet been saved to the local machine.

Example:

```
MailAtt maAtt;
MailMsg mmMsg;
BOOL bDownload = TRUE;
if (MailFirst(hMsgStore, &mmMsg))
{
    // see if there are un-retrieved attachments
    if (mmMsg.dwFlags & MAIL_STATUS_ATT_REQ)
    {
        MailRequestAttachment(hMsgStore, &mmMsg, &maAtt,
bDownload);
    }
}
```

Where:

- `hMsgStore` is an open handle to the message store.
- `mmMsg` is a `MailMsg` structure specifying the mail message whose attachment you want to download.
- `maAttachment` is a `MailAtt` structure the function will use to return the attachment.
- `bDownload` is a `BOOL` value specifying whether to download the attachment (`TRUE`) or cancel a previous request (`FALSE`) to download the attachment.

MailSetField()

```
#include <msgstore.h>
```

Sets a specified header field from the mail message.

Example:

```
LPWSTR szFieldName = TEXT("Subject");
LPWSTR szVal = TEXT("A New Subject!");
MailSetField(&mmMsg, szFieldName, szVal);
```

Where:

- `mmMsg` is a `MailMessage` previously retrieved with a call to `MailFirst()`, `MailGet()`, etc.

- `szFieldName` is the name of the mail message field to retrieve. Possible values include

  "Subject" Retrieves the subject field of the message.

  "To" Retrieves the recipient field of the message.

  "From" Retrieves the sender field of the message.

  "CC" Retrieves the carbon-copy-recipient field...or any custom header you'd like to add to the message.

- `szVal` is a string specifying a value for the header field. If this parameter is NULL, the header field will be deleted.

MailSetSort()

```
#include <msgstore.h>
```

Sets the sort order for the messages of an open message store handle.

Example:

```
MAILSORTFIELD msfSort = MAIL_SORT_DATE;
MailSetSort(hMsgStore, &msfSort)
```

Where:

- `hMsgStore` is an open handle to the message store.
- `msfSort` is a `MAILSORTFIELD` value specifying how the messages are to be sorted.

The `MAILSORTFIELD` is an enumeration defined as follows:

```
typedef enum {
MAIL_SORT_FROM,
MAIL_SORT_SUBJ,
MAIL_SORT_DATE,
MAIL_SORT_SIZE
} MAILSORTFIELD;
```

MailUpdate()

```
#include <msgstore.h>
```

Updates an existing mail message entry in the message store.

Example:

```
BOOL bUpdateAllFields = TRUE;
MailUpdate(hMsgStore, &mmMsg, bUpdateAllFields);
```

Where:

- `hMsgStore` is an open handle to the message store.
- `mmMsg` is a `MailMessage` structure that contains the updated information.
- `bUpdateAllFields` is a `BOOL` specifying whether all of the existing message's fields should be overwritten (`TRUE`) or just those specified in the `dwFlags` member of `mmMsg`.

ModifyAddressCard()

```
#include <addrstor.h>
```

Saves changes to an address card back to the Contacts database.

Example:

```
AddressCard ac;
CEOID oid;
int iIndex;
//...
//obtain oid of card to modify...then
//populate AddressCard ac
//...
ModifyAddressCard(&ac, oid,  &iIndex);
```

Where:

- `ac` is an `AddressCard` structure containing the values to be written back to the Contacts database.
- `oid` is the `CEOID` of the card (record) to modify.
- `iIndex` is an integer the function will use to return the index of the modified record.

The AddressCard structure is defined as follows:

```
typedef struct _AddressCard {
SYSTEMTIME stBirthday;
SYSTEMTIME stAnniversary;
TCHAR *pszBusinessFax;
TCHAR *pszCompany;
TCHAR *pszDepartment;
TCHAR *pszEmail;
TCHAR *pszMobilePhone;
TCHAR *pszOfficeLocation;
TCHAR *pszPager;
TCHAR *pszWorkPhone;
TCHAR *pszTitle;
TCHAR *pszHomePhone;
TCHAR *pszEmail2;
TCHAR *pszSpouse;
TCHAR *pszNotes;
TCHAR *pszEmail3;
TCHAR *pszHomePhone2;
TCHAR *pszHomeFax;
TCHAR *pszCarPhone;
TCHAR *pszAssistant;
TCHAR *pszAssistantPhone;
TCHAR *pszChildren;
TCHAR *pszCategory;
TCHAR *pszWebPage;
TCHAR *pszWorkPhone2;
TCHAR *pszNamePrefix;
TCHAR *pszGivenName;
TCHAR *pszMiddleName;
TCHAR *pszSurname;
TCHAR *pszGeneration;
TCHAR *pszHomeAddrStreet;
TCHAR *pszHomeAddrCity;
TCHAR *pszHomeAddrState;
TCHAR *pszHomeAddrPostalCode;
TCHAR *pszHomeAddrCountry;
TCHAR *pszOtherAddrStreet;
TCHAR *pszOtherAddrCity;
TCHAR *pszOtherAddrState;
TCHAR *pszOtherAddrPostalCode;
```

```
TCHAR *pszOtherAddrCountry;
TCHAR *pszOfficeAddrStreet;
TCHAR *pszOfficeAddrCity;
TCHAR *pszOfficeAddrState;
TCHAR *pszOfficeAddrPostalCode;
TCHAR *pszOfficeAddrCountry;
BYTE *rgbReserved[84]
} AddressCard;
```

OpenAddressBook()

```
#include <addrstor.h>
```

Opens the Address Book; returns a nonzero value if successful.

Example:

```
HANDLE hWnd = NULL;
HHPRTAG hptProp = HHPR_FILEAS;
OpenAddressBook(hWnd, hptProp);
```

Where:

- `hWnd` is a handle to a window or NULL. Uses a window handle to receive notification if the Contacts database is modified while your application is accessing it.
- `hptProp` is an `HHPRTAG` value specifying the sort order to apply to the Contacts database while in use by your application.

OpenAddressCard()

```
#include <addrstor.h>
```

Retrieves all properties of an address card, given the card's `CEOID`.

Example:

```
ULONG uFlag = OAC_ALLOCATE;
OpenAddressCard (oidContact, &ac, uFlag);
```

Where:

- `oidContact` is the `CEOID` of the address card whose properties the function is to retrieve. The `CEOID` of the card must be known ahead of time; you can get the `CEOID` of a card by calling functions like `FindFirstEntry()` and `GetMatchingEntry()`.

- ac is an AddressCard structure that the function uses to return the address card's properties.
- uFlag is a ULONG value that can be one of the following values:

  OAC_ALLOCATE Specifies that the function should allocate additional memory to accommodate any possible modifications to string properties.

  0 Specifies that the function should not allocate the additional memory and essentially makes the card read-only.

Random()

```
#include <winbase.h>
```

Returns a DWORD random number.

Example:

```
DWORD dwRandom;
dwRandom = Random();
```

Where:

- dwRandom is a DWORD used to store the return value of the function.

NOTE This function takes no parameters.

RapiFreeBuffer()

```
#include <rapi.h>
```

Frees memory allocated by the RAPI functions CeFindAllDatabases(), CeFindAllFiles(), and CeReadRecordProps().

Example:

```
dwType = 0;
LPCEDB_FIND_DATA pFindData;
wFlags = FAD_OID | FAD_FLAGS | FAD_NAME | FAD_TYPE FAD_NUM_RECORDS
| FAD_NUM_SORT_ORDER | FAD_SORT_SPECS;
CeFindAllDatabases( DbType, 0xFF, &cFound, &pFindData);
//...
```

```
//do something with found databases...
//then, when finished....
RapiFreeBuffer(pFindData);
```

Where:

- `pFindData` is a buffer allocated by one of the RAPI functions listed above; in this case, `CeFindAllDatabases()`.

RasGetEntryDevConfig()

Although still provided for backward compatibility, this function has been replaced by `RasGetEntryProperties()`, which is supported on all versions of Windows, including 98 and NT.

RasSetEntryDevConfig()

Although still provided for backward compatibility, this function has been replaced by `RasSetEntryProperties()`, which is supported on all versions of Windows, including 98 and NT.

RecountCards()

```
#include <addrstor.h>
```

Causes the system to recount the number of cards in the Contacts database. Useful if your application is set up to receive notification when the Contacts database is modified while your application is using it.

Example:

```
RecountCards();
```

NOTE This function has no parameters and does not return a value.

RegisterDevice()

```
#include <winbase.h>
```

Registers a hardware device.

Example:

```
HANDLE hDev;
DWORD dwIndex;
DWORD dwInstanceInfo; //depends on device being registered
TCHAR szDevType[MAX_PATH] = TEXT("WAV");
TCHAR szDriverName[MAX_PATH] = TEXT("snd.dll");
dwIndex = 2;
hDev = RegisterDevice(szDevType, dwIndex, szDriverName,
dwInstanceInfo);
```

Where:

- `hDev` is a handle to the device if the function is successful.
- `szDevType` is a string identifying the type of device.
- `dwIndex` is a `DWORD` specifying the index of the device.
- `szDriverName` is a string containing the name of the driver for the device being registered.
- `dwInstanceInfo` is a `DWORD` specifying the instance information of the device. For example, this might be a hexidecimal memory address.

SetColumnProperties()

```
#include <addrstor.h>
```

Sets the available sort orders for the Contacts database.

Example:

```
HHPRTAG hptProps[4]; //max. 4 properties to sort on
hptProps[0] = HHPR_SURNAME;
int iColumnCount = 1;
SetColumnProperties(&hptProps, iColumnCount);
```

Where:

- `hptProps` is an array of `HHPRTAG` values, specifying the properties on which the Contacts database is to be sorted.
- `iColumnCount` specifies the number of positions in the array containing valid `HHPRTAG` values. The maximum value for this parameter is 4.

SetDaylightTime()

```
#include <winbase.h>
```

Specifies to the CE operating system whether or not daylight saving time is in effect.

Example:

```
DWORD dwDaylight = 1;
SetDaylightTime(dwDaylight);
```

Where:

- `dwDaylight` is a `DWORD` value specifying:
  - `1` daylight saving time is currently in effect.
  - `0` daylight saving time is not currently in effect.

SetMask()

```
#include <addrstor.h>
```

Indicates which of the properties in an AddressCard structure are being used by the application.

Example:

```
AddressCard ac;
HHPRTAG hptTag;
memset (&ac, 0, sizeof (AddressCard));
ac.pszGivenName = TEXT("Karen");
ac.pszSurname = TEXT("Johnson");
ac.pszCompany = TEXT("Widget Co.");
ac.pszEmail = TEXT("Karen@widgetco.com");
// tell system which properties we are setting.
hptTag = HHPR_GIVEN_NAME;
SetMask (&ac, hptTag);
hptTag = HHPR_SURNAME;
SetMask (&ac, hptTag);
hptTag = HHPR_COMPANY_NAME;
SetMask (&ac, hptTag);
hptTag = HHPR_EMAIL1_EMAIL_ADDRESS
SetMask (&ac, hptTag);
```

Where:

- ac is an AddressCard structure.
- hptTag is an HHPRTAG value specifying which members of the AddressCard structure you're using.

SetSortOrder()

```
#include <addrstor.h>
```

Selects one of the available sort orders and marks it as active. The sort order selected must be one of the sort orders specified with a call to OpenAddressBook() or SetColumnProperties().

Example:

```
HHPRTAG hptProps[4]; //max. 4 properties to sort on
HHPRTAG hptProp;
hptProps[0] = HHPR_SURNAME;
int iColumnCount = 1;
SetColumnProperties(&hptProps, iColumnCount);
hptProp = HHPR_SURNAME;
SetSortOrder(hptProp);
```

Where:

- hptProp is an HHPRTAG value specifying the sort order to use.

SHCreateShortcut()

```
#include <shellapi.h>
```

Creates a shortcut.

Example:

```
TCHAR szShortcutFile[MAX_PATH];
TCHAR szActualExe[MAX_PATH];
szShortcutFile = TEXT("\\MyApp.lnk");
szActualExe = TEXT("\\Windows\\MyApp.exe");
SHCreateShortcut(szShortcutFile, szActualExe);
```

Where:

- `szShortcutFile` is a string specifying the full path of the shortcut file.
- `szActualExe` is a string specifying the full path to the actual executable file.

SHGetShortcutTarget()

```
#include <shellapi.h>
```

Retrieves the path to the target executable of the specified shortcut.

Example:

```
TCHAR szShortcutFile[MAX_PATH];
TCHAR szActualExe[MAX_PATH];
int iActualExeLen = MAX_PATH;
szShortcutFile = TEXT("\\MyApp.lnk");
SHGetShortcutTarget(szShortcutFile, szActualExe, iActualExeLen);
```

Where:

- `szShortcutFile` is a string specifying the full path of the shortcut file.
- `szActualExe` is a string the function uses to return the full path to the actual executable file.
- `iActualExeLen` is an integer specifying the maximum possible length of `szActualExe`.

SHLoadDIBitmap()

```
#include <shellapi.h>
```

Loads a bitmap from a file and returns a handle to that bitmap if successful.

Example:

```
HBITMAP hBmp;
TCHAR szBmpName[MAX_PATH] = TEXT("\\MyApp\\resources\\logo.bmp");
hBmp = SHLoadDIBitmap(szBmpName);
```

Where:

- `hBmp` is the handle of the bitmap returned by the function if the call is successful.
- `szBmpName` is a string specifying the name of the bitmap file.

SHShowOutOfMemory()

```
#include <shellapi.h>
```

Shows the Out of Memory message.

Example:

```
SHShowOutOfMemory(hWnd, 0);
```

Where:

- `hWnd` is the handle of the window that will own the dialog.

NOTE The second parameter of this function is always 0.

TIP It is recommended that you use this function if your application runs low on memory. This function will work under low-memory conditions, whereas creating your own new dialog box may fail under low-memory conditions.

TransparentImage()

```
#include <wingdi.h>
```

Copies the contents of one HDC to another, with the ability to specify that one of the colors in the source HDC is to be considered a transparent color.

Example:

```
HDC hdcDest, hdcSource;
COLORREF clrTrans;
LONG DstX, DstY, DstCx, DstCy;
LONG SrcX, SrcY, SrcCx, SrcCy;
//...
//set up hdcDest and hdcSource...
//...
DstX = 0;
DstY = 0;
DstCx = 16;
DstCy = 16;
SrcX = 0;
SrcY = 0;
```

```
    SrcCx = 16;
    SrcCy = 16;
    clrTrans = 0x00FFFFFF; //white is the transparent color
    TransparentImage(hdcDest,DstX, DstY, DstCx, DstCy, hdcSource, SrcX,
SrcY, SrcCx, SrcCy, clrTrans);
```

Where:

- `hdcDest` is the destination HDC.
- `DstX` is a `LONG` integer specifying the destination *x* coordinate.
- `DstY` is a `LONG` integer specifying the destination *x* coordinate.
- `DstCx` is a `LONG` integer specifying the destination *x* coordinate.
- `DstCy` is a `LONG` integer specifying the destination *x* coordinate.
- `hdcSource` is the source HDC.
- `SrcX` is a `LONG` integer specifying the destination *x* coordinate.
- `SrcY` is a `LONG` integer specifying the destination *x* coordinate.
- `SrcCx` is a `LONG` integer specifying the destination *x* coordinate.
- `SrcCy` is a `LONG` integer specifying the destination *x* coordinate.
- `clrTrans` is a `COLORREF` value specifying the transparent color. A `COLORREF` is a 32-bit value specifying the RGB value for a color. `COLORREF`s have the form `0x00rrggbb`, where `rr` is the red component, `gg` is the green component, and `bb` is the blue component.

TIP Essentially, `TransparentImage()` is a `BitBlt()` that supports transparency.

*Var*YYYY*From*ZZZZ() Family of Functions

```
#include <oleauto.h>
```

All of the functions having the form `Var`*YYYY*`From`*XXXX*`()` are part of the OLE automation of Windows CE. Instead of listing each individual function here (there are approximately 170 different functions), we will provide a description of the general form and detail the different data types involved.

Each of these `VarYYYYFromXXXX()` functions converts from a variant type to a nonvariant type. Table B.1 shows the abbreviations for each of these data types and their meanings.

TABLE B.1: Variant Data Types

Abbreviation	Data Type
Bool	Boolean
Bstr	BSTR
Cy	currency
Date	date
Dec	decimal
Disp	IDispatch
R4	float
R8	double
I1	char
I2	short
I4	long
I8	double
Str	char*
UI1	byte
UI2	unsigned short
UI4	unsigned long

A full listing of all of the conversion functions follows:

- `VarBoolFromCy()`
- `VarBoolFromDate()`
- `VarBoolFromDec()`

- VarBoolFromDisp()
- VarBoolFromI1()
- VarBoolFromI2()
- VarBoolFromI4()
- VarBoolFromR4()
- VarBoolFromR8()
- VarBoolFromStr()
- VarBoolFromUI1()
- VarBoolFromUI2()
- VarBoolFromUI4()
- VarBstrFromBool()
- VarBstrFromCy()
- VarBstrFromDate()
- VarBstrFromDec()
- VarBstrFromDisp()
- VarBstrFromI1()
- VarBstrFromI2()
- VarBstrFromI4()
- VarBstrFromR4()
- VarBstrFromR8()
- VarBstrFromUI1()
- VarBstrFromUI2()
- VarBstrFromUI4()
- VarCyFromBool()
- VarCyFromDate()
- VarCyFromDec()

- VarCyFromDisp()
- VarCyFromI1()
- VarCyFromI2()
- VarCyFromI4()
- VarCyFromR4()
- VarCyFromR8()
- VarCyFromStr()
- VarCyFromUI1()
- VarCyFromUI2()
- VarCyFromUI4()
- VarDateFromBool()
- VarDateFromCy()
- VarDateFromDec()
- VarDateFromDisp()
- VarDateFromI1()
- VarDateFromI2()
- VarDateFromI4()
- VarDateFromR4()
- VarDateFromR8()
- VarDateFromStr()
- VarDateFromUdate()
- VarDateFromUI1()
- VarDateFromUI2()
- VarDateFromUI4()
- VarDecFromBool()
- VarDecFromCy()

- VarDecFromDate()
- VarDecFromDisp()
- VarDecFromI1()
- VarDecFromI2()
- VarDecFromI4()
- VarDecFromR4()
- VarDecFromR8()
- VarDecFromStr()
- VarDecFromUI1()
- VarDecFromUI2()
- VarDecFromUI4()
- VarI1FromBool()
- VarI1FromCy()
- VarI1FromDate()
- VarI1FromDec()
- VarI1FromDisp()
- VarI1FromI2()
- VarI1FromI4()
- VarI1FromR4()
- VarI1FromR8()
- VarI1FromStr()
- VarI1FromUI1()
- VarI1FromUI2()
- VarI1FromUI4()
- VarI2FromBool()
- VarI2FromCy()

- VarI2FromDate()
- VarI2FromDec()
- VarI2FromDisp()
- VarI2FromI1()
- VarI2FromI4()
- VarI2FromR4()
- VarI2FromR8()
- VarI2FromStr()
- VarI2FromUI1()
- VarI2FromUI2()
- VarI2FromUI4()
- VarI4FromBool()
- VarI4FromCy()
- VarI4FromDate()
- VarI4FromDec()
- VarI4FromDisp()
- VarI4FromI1()
- VarI4FromI2()
- VarI4FromR4()
- VarI4FromR8()
- VarI4FromStr()
- VarI4FromUI1()
- VarI4FromUI2()
- VarI4FromUI4()
- VarR4FromBool()
- VarR4FromCy()

- VarR4FromDate()
- VarR4FromDec()
- VarR4FromDisp()
- VarR4FromI1()
- VarR4FromI2()
- VarR4FromI4()
- VarR4FromR8()
- VarR4FromStr()
- VarR4FromUI1()
- VarR4FromUI2()
- VarR4FromUI4()
- VarR8FromBool()
- VarR8FromCy()
- VarR8FromDate()
- VarR8FromDec()
- VarR8FromDisp()
- VarR8FromI1()
- VarR8FromI2()
- VarR8FromI4()
- VarR8FromR4()
- VarR8FromStr()
- VarR8FromUI1()
- VarR8FromUI2()
- VarR8FromUI4()
- VarUI1FromBool()
- VarUI1FromCy()

- VarUI1FromDate()
- VarUI1FromDec()
- VarUI1FromDisp()
- VarUI1FromI1()
- VarUI1FromI2()
- VarUI1FromI4()
- VarUI1FromR4()
- VarUI1FromR8()
- VarUI1FromStr()
- VarUI1FromUI2()
- VarUI1FromUI4()
- VarUI2FromBool()
- VarUI2FromCy()
- VarUI2FromDate()
- VarUI2FromDec()
- VarUI2FromDisp()
- VarUI2FromI1()
- VarUI2FromI2()
- VarUI2FromI4()
- VarUI2FromR4()
- VarUI2FromR8()
- VarUI2FromStr()
- VarUI2FromUI1()
- VarUI2FromUI4()
- VarUI4FromBool()
- VarUI4FromCy()

- VarUI4FromDate()
- VarUI4FromDec()
- VarUI4FromDisp()
- VarUI4FromI1()
- VarUI4FromI2()
- VarUI4FromI4()
- VarUI4FromR4()
- VarUI4FromR8()
- VarUI4FromStr()
- VarUI4FromUI1()
- VarUI4FromUI2()

INDEX

Note to the Reader: Throughout this index **boldfaced** page numbers indicate primary discussions of a topic. *Italicized* page numbers indicate illustrations.

A

abs(), **450**
accelerators, **425**
accessories, **36–38**
acos(), **450**
Active Sync, **343**
ActiveX Control Pack
  Common Dialog, **234–235**, *235*
  grid, **231–232**, *232*
  ImageList, **234**
  ListView, **234**
  overview, **231**
  solving read-only problem, **232–233**
  tab strip, **233–234**, *234*
  TreeView, **234**
adding
  Add button, **212–213**, *212*
  Add/Insert dialog, **214**, *214*
  AddAddressCard(), **474–475**
  AddRecord(), **180**, **185–186**
  EditAdd Record, **135–136**, *135*, *136*
Address Book API, **419–420**
allocating memory
  AreaCodeDatabaseRead(), **137**
  calloc(), **77**
  checking return results of, **57**
  EXE file size, **56–57**
  LMEM_ZEROINIT, **77**
  LocalAlloc(), **58–60**, 77, 84
  mass, **58–59**
  overview, **56**
  types of, **59–60**
Alt+Enter for property sheets, **422**
Alt+letter for underlined accelerators, **425**
Alt+tap for pop-up menus, **422**
Alt-click combination for right-click functionality, **8**
Alt-tap behavior, **78**
ANSI
  ANSI-based RTL functions, **47**, *47*
  ANSI-specific string type mappings, **44**, *44*
  converting between Unicode- and ANSI-based text files, **49–51**, *49*, *51*
  vs. Unicode files, **97**
API
  Address Book, **419–420**
  vs. CCeDBDatabase methods, **180–182**
  functions for Windows CE 2.0. *See* API functions for Windows CE 2.0
  msgstore, **420**
  Polyline(), **78**
  reduction of, **7**
  Remote API. *See* RAPI (Remote API)
  replacing GetDesktopWindow(), **164–165**
  SetFilePointer, **108–109**
  structured storage
    AreaCodeDatabaseDelete(), **140**
    AreaCodeDatabaseFinish(), **140**
    AreaCodeDatabaseInsert(), **139–140**
    AreaCodeDatabaseRead(), **136–139**
  substituting for missing functions
    calloc(), **77**
    MoveToEx()/LineTo(), **77–78**
    WM_RBUTTONDOWN, **78–79**
  Unicode text and, **43**
API functions for Windows CE 2.0
  AddAddressCard(), **474–475**
  BatteryNotifyOfTimeChange(), **476**
  CeCheckPassword(), **476–477**
  CeClearUserNotification(), **477**
  CeCloseHandle(), 289, **477**
  CeCopyFile(), **477–478**
  CeCreateDatabase, 122, 292, **478–479**
  CeCreateDirectory(), **479**
  CeCreateFile(), 289, **479–480**
  CeCreateProcess(), 292, **480–481**
  CeDeleteDatabase(), 180, 292, **481–482**
  CeDeleteFile(), 289, **482**
  CeDeleteRecord(), 140, 292, **482**
  CeFindAllDatabases(), **482–483**

CeFindAllFiles(), **484**
CeFindClose(), 288, **484–485**
CeFindNextDatabase(), 176, 292, **486**
CeFindNextFile(), 288, **486–487**
CeGetClassName(), 293, **487**
CeGetDesktopDeviceCaps(), 285, 286, *286*, **487–488**
CeGetDeviceId(), **488**
CeGetFileAttributes(), **489**
CeGetFileSize(), **489**
CeGetFileTime(), **490**
CeGetLastError(), **490–491**
CeGetSelectedDeviceId(), **491**
CeGetSpecialFolderPath(), 289, 291, **491**
CeGetStoreInformation(), **492**
CeGetSystemInfo(), 285, **492**
CeGetSystemMetrics(), **493**
CeGetSystemPowerStatusEx(), 285, **493–494**
CeGetTempPath(), 289, **494**
CeGetUserNotificationPreferences(), **494–495**
CeGetVersionEx(), 285, **495–496**
CeGetVersionInfo(), **285**
CeGetWindow(), 292, **496**
CeGetWindowText(), 292, **497**
CeGlobalMemoryStatus(), 285, 286, **497**
CeHandleAppNotifications(), **498**
CeMoveFile(), **498**, **498**
CeOIDGetEx(), 285
CeOidGetInfo(), 180, **498–499**
CeOpenDatabase(), 128, 137, 180, 181, 292, **499**
CeRapiFreeBuffer(), **500**
CeRapiGetError(), **500**
CeRapiInit(), 283–284, 293, **500–501**
CeRapiInitEx(), **501**
CeRapiInvoke(), **501–502**
CeRapiUninit(), 283–284, **502**
CeReadFile(), 289, **502–503**
CeReadRecordProps(), 136, 138, 292, **503–504**
CeRegCloseKey(), 287, **504**
CeRegCreateKeyEx(), 287, **504–505**
CeRegDeleteKey(), 287, **505**
CeRegDeleteValue(), 287, **506**
CeRegEnumKeyEx(), 287, **506–507**
CeRegEnumValue(), 287, **507–508**
CeRegOpenKeyEx(), 287, **508–509**
CeRegQueryInfoKey(), 287, **509–510**
CeRegQueryValueEx(), 287, **510–511**
CeRegSetValueEx(), 287, **511–512**
CeRemoveDirectory(), **512**
CeRunAppAtEvent(), **512–513**
CeRunAppAtTime(), **513**
CeSeekDatabase(), 131, 132, 133, 134, 135, 180, 181, 292, **513–514**
CeSetDatabaseInfo(), 182, 292, **514–515**
CeSetEndOfFile(), **515**
CeSetFileAttributes(), **515–516**
CeSetFilePointer(), **516**
CeSetFileTime(), **516–517**
CeSetUserNotification(), **517–518**
CeSHCreateShortcut(), 292, **518–519**
CeSHGetShortcutTarget(), 292, **519**
CeWriteFile(), 289, **519–520**
CeWriteRecordProps(), 139–140, 292, **520–521**
CloseAddressBook(), **521**
CommandBands_AddAdornments(), **521–522**
CommandBands_AddBands(), **522–523**
CommandBands_Create(), **523–524**
CommandBands_GetCommandBar(), **524**
CommandBands_GetRestoreInformation(), **525**
CommandBands_Height(), **525**
CommandBands_IsInvisible(), **526**
CommandBands_Show(), **526**
CommandBar_AddAdornments(), **526–527**
CommandBar_AddButtons(), 527–528
CommandBar_AddToolTips(), **528**
CommandBar_Create(), **528–529**
CommandBar_DrawMenuBar(), **529**
CommandBar_GetMenu(), **529**
CommandBar_Height(), **529–530**
CommandBar_InsertButton(), **530**
CommandBar_InsertComboBox(), **530–531**
CommandBar_InsertMenubar(), **531**
CommandBar_InsertMenubarEx(), **531–532**
CommandBar_IsVisible(), **532**
CommandBar_Show(), **532**
CreateAddressBook(), **533**
CreateFileForMapping(), **533–534**
DeleteAddressCard(), **534**
DeleteAndRenameFile(), **534–535**
DeregisterDevice(), **535**
EnableEUDC(), **535–536**
EnableHardwareKeyboard(), **536**
FindFirstEntry(), **536–539**
FreeAddressCard(), **539–540**

FtpCommand(), 540–541
GetAddressCardIndex(), 541
GetAddressCardOid(), 542
GetAddressCardProperties(), 542
GetClipboardDataAlloc(), 543
GetColumnProperties(), 543
GetMatchingEntry(), 544
GetMessageSource(), 544–545
GetMouseMovePoints(), 545
GetNumberOfAddressCards(), 546
GetPropertyDataStruct(), 546
GetSortOrder(), 547
GetStoreInformation(), 547
GetSystemPowerStatusEx(), 548
ImageList_Duplicate(), 548–549
InitCommonControlsEx(), 549–550
InitHTMLControl(), 550
InitInkX(), 550
InterlockedTestExchange(), 551
MailClose(), 551
MailDelete, 551–552
MailDeleteAttachment(), 552–553
MailErrorMsg(), 553–554
MailFirst(), 554–555
MailFree(), 555
MailGet(), 555–556
MailGetAttachment(), 556
MailGetField(), 557
MailGetFolderId(), 557–558
MailGetFolderName(), 558
MailGetSort(), 558–559
MailGetSvcId(), 559
MailHeaderLen(), 560
MailLocalAttachmentLen(), 560
MailNext(), 560
MailOpen(), 561
MailOpenNotify(), 561
MailPut(), 562
MailPutAttachment(), 562–563
MailPutFolder(), 563
MailRequestAttachment(), 563–564
MailSetField(), 564–565
MailSetSort(), 565
MailUpdate(), 566
ModifyAddressCard(), 566–568
OpenAddressBook(), 568
OpenAddressCard(), 568–569
Random(), 569
RapiFreeBuffer(), 569–570
RasGetEntryDevConfig(), 570
RasSetEntryDevConfig(), 570
RecountCards(), 570
RegisterDevice(), 570–571
SetColumnProperties(), 571
SetDaylightTime(), 572
SetMask(), 572–573
SetSortOrder(), 573
SHCreateShortcut(), 573–574
SHGetShortcutTarget(), 574
SHLoadDIBitmap(), 574
SHShowOutOfMemory(), 575
TransparentImage(), 575–576
VarYYYYFromZZZZ() functions, 576–584
Application Manager
creating INI files, **398–399**, *398*
creating setup programs, **399–402**
Logo program installation, **411**, 412
application templates, **226–227**, *227*
area code lookup utility
AreaCodeDataBaseCreate(), **125–126**
AreaCodeDatabaseDelete(), **140**
AreaCodeDatabaseFinish(), **140**
AreaCodeDatabaseInsert(), **139–140**
AreaCodeDatabaseRead(), **136–139**
Edit/Add Record dialog, **135–135**, *135*, *136*
overview, **135–135**, *135*, *136*
array declarations, **256–260**, *257*
ASA (Adaptive Server Anywhere)
administration and management, **343**
command line options, **344**
comparison of features, **356–358**
configuring application, **344**
copying data to desktop, **357**
Customer List report, **345–346**, *345*
data compression, **357**
database schema, **342–343**
demo, **344–347**, *345*
Login dialog, **344–345**, *345*
overview, **341**
proc_name, **343**
Report dialog, **345**, *345*
report_id, **343**
report_name, **343**
Sales Report, **346**

sample application, **342–347**, *345*
security_level, **343**
SQL support, **357**
storage requirements, **357**
stored procedures, **357**
synching data with desktop, **357**
user privileges, **358**
asademo.db, **344**
atan(), **450**
atan2(), **451**
ATM conversion,
array declarations, **256–260**, *257*
Bitmaps, **250**
change Global to Public, **249**
country and bit maps, **266–269**
Declare statements, **246–247**
eliminating code, **274–275**
eliminating forms, **270–274**, *271*
eliminating optional controls and DLLs, **274**
Explicit Type declarations, **249**
Explicit Types, **249**
icons, **250**
Image Control, **250**
mechanical issues, **249–261**, *257*
modal forms, 248, **253–255**
modifying resource loads, **255–260**, *257*
mouse/cursor code, **252–253**
optimizing, **270–275**
optional features, **262–269**, *264*, *265*
overview, **244–249**, *245*, *246*, *276*, *276*
planning porting process, **247–249**
porting from VB to VBCE, **247**
preparing frmAmountWithdrawn Form, **261**
sound, **263–265**, *264*, *265*
Typecasts, **250**
using Developer Studio, **256–257**, *257*
WAV files, **250–253**, *262–266*, *264*, *265*
atoi(), **451**
atol(), **451**
AutoPC SDK, **69**

B

bar code scanners, 38, **363**, *365*
BatteryNotifyOfTimeChange(), **476**
Bin paths, **147–148**
bitmaps, 250, **266–269**
BLOB (Binary Large Object) data, **118**, 297
buttons
Add button, **212–213**, *212*
Close "X" button, 25, 204, **418**
displaying open documents, **432**
Help "?" button, **416**
HPC applications, **419**
InsertButtons(), **204–206**
Logo program
taskbar buttons, **432**
UI requirements, **416**, 417, 419, 424, 425
WM_BUTTONDOWN, **78–79**
WM_BUTTONUP, **423**

C

C runtime library functions
abs(), **450**
acos(), **450**
asin(), **450**
atan(), **450**
atan2(), **451**
atoi(), **451**
atol(), **451**
_cabs(), **438**
_chsign(), **438**
_clearfp(), **439**
_controlfp(), **439**
_copysign(), **439**
cos(), **451**
cosh(), **452**
difftime(), **452**
div(), **452**
_evct(), **439**
exp(), **452**
fabs(), **453**
_fcvt(), **439**
_finite(), **440**
floor(), **453**
fmod(), **453**
_fpclass(), **440**
fpieee_flt(), **440**
_fpreset(), **440**
free(), **453**
frexp(), **453**

_gcvt(), 441
_hypot(), 441
_isnan(), 441
iswalnum(), 454
iswalpha(), 454
iswascii(), 454
iswcntrl(), 454
iswctype(), 455
iswgraph(), 455
iswlower(), 455
iswprint(), 455
iswpunct(), 456
iswspace(), 456
iswupper(), 456
iswxdigit(), 456
_itoa(), 441
_itow(), 441
_j0(), 442
_j1(), 442
_jn(), 442
labs(), 457
ldexp(), 457
ldiv(), 457
log(), 457
log10(), 458
_logb(), 442
longjmp(), 458
_lrotl(), 442
_lrotr(), 443
_ltoa(), 443
_ltow(), 443
malloc(), 458
_matherr(), 443
mbstowcs(), 458
_memccpy(), 444
memchr(), 459
memcmp(), 459
memcpy(), 459
_memicmp(), 444
memmove(), 459
memset(), 460
modf(), 460
_msize(), 444
_nextafter(), 444–445
pow(), 460
qsort(), 460
rand(), 460–461
realloc(), 461
_rotl(), 445
_rotr(), 445
_scalb(), 445
sin(), 461
sinh(), 461
sqrt(), 461
srand(), 462
_statusfp(), 445
strcat(), 462
strchr(), 462
strcmp(), 462
strcpy(), 462
strcspn(), 463
strlen(), 463
strncat(), 463
strncmp(), 463
strncpy, 463–464
strstr(), 464
strtok(), 464
_swab(), 446
swprintf(), 464
swscan(), 465
tan(), 465
tanh(), 465
towlower(), 465
towupper(), 466
_ultoa(), 446
_ultow(), 446
vswprintf(), 466
wcscat(), 466
wcschr(), 467
wcscmp(), 467
wcscpy(), 467
wcscspn(), 467
_wcsdup(), 446
_wcsicmp(), 447
wcslen(), 468
_wcslwr(), 447
wcsncat(), 468
wcsncmp(), 468
wcsncpy(), 468–469
_wcsnicmp(), 447
_wcsnset(), 447–448
wcspbrk(), 469
wcsrchr(), 469
_wcsrev(), 448

_wcsset(), **448**
wcsspn(), **469–470**
wcsstr(), **470**
wcstod(), **470**
wcstok(), **470**
wcstol(), **471**
wcstombs(), **471**
wcstoul(), **471**
_wcsupr(), **448**
wsprintf(), **471**
_wtoi(), **449**
_wtol(), **449**
_y0(), **449**
_y1(), **449**
_yn(), **450**
CAB files Logo program installation, **411**, 412, 413
Cab Wizard
creating Cab files
Cab archive, **397–398**
INF files, **392–397**
INI files, **398–399**, *398*
creating setup programs, **399–402**
overview, **391–392**
predefined destination directories, **393–394**
_cabs(), **438**
Canon PowerShot A5, 368
CapEdit control
Handheld PC/Pro devices (HPC/Pros), **34**
Handheld PCs (HPCs), **27**
Palm-size PCs (PPCs), **23–24**
Carcalc, **221–222**, *221*
case studies
choosing development machine
conclusion, **372**
cost analysis, **372**
HPC devices, **371**
HPC/Pro devices, **371**
overview, **370**
PPC devices, **370–371**
solution, **370**
insurance field agents
conclusion, **370**
cost analysis, **368–369**, *369*
digital cameras, **367**
HPC/Pro, **368**
HPC/Pro-based cost analysis, **369**, *369*
lap-top based cost analysis, **368–369**, *369*
overview, **366–367**
solution, **367**
technical issues, **367–368**
Inventory Management System
conclusion, **366**
cost, **365–366**
overview, **362**
technical issues, **362–364**
Casio
PA-2400 "tablet" device, **35**, *35*
PA-2500 Handheld PC (HPC), **25**, *26*
Vertical Markets division, **58**
C/C++ language for CE
calloc(), **77**
changing program logic, **79–80**
CreateFile(), **80–81**
CreateProcess(), **81**
exception handling methods, **79–80**
fclose(), **88–89**
fgetc(), **89–90**
fgets(), **92–94**
FILE*, **81–83**
fopen(), **83–88**
fprint(), **97–99**
fputc(), **90–91**
fputs(), **94–95**
fread(), **95–96**
fscanf(), **99–107**
fseek(), **108–109**
fwrite(), **96–97**
MoveToEx()/Line To(), **77–78**
overview, **76–77**
Readfile(), **80**
swscanf(), **99–107**
try..catch, **79–80**
Unicode and file access, 95
Win32 API file-access functions, **80–81**
WM_RBUTTONDOWN, **78–79**
wrapper functions, **81**
Writefile(), **80**
wstrlen(), **95**
CCeDBDatabase, 174, **179–182**, *180–182*
CCeDBEnum, 174, **176**
CCeDBProp, 174, **176–178**, *177*, 186
CCeDBRecord, 174, **179**, *179*
CCeSocket, 174, **175–176**, *175*
CDC, **192**

CE communications
  hardware aspects
    IR ports, **307**
    modems, **307**
    overview, **306**
    PCMCIA cards, **307**
    serial ports, **307**
  overview, **306**
  PC cards/PCMCIA cards
    application start, **318–320**, *320*
    card changes, **321–326**, *323*
    overview, **318**
  PC cards/PCMCIA modems, **318**
  software aspects
    built-in modems, **314–317**, *318*
    Infrared Sockets (IrSock), **314**, 332
    IR communications, **308–314**, *313*
    IrCOMM, **312–314**, *313*
    modem-based communications, **314–318**, *318*
    overview, **308**
    Raw IR, **309–312**
    serial communications, **308–309**
    standard external modems, **314**
  Winsock-based
    Infrared Sockets (IrSock), 314, **332**
    overview, **327**
    Winsock 1.1, **327–332**, *328*
CE devices. *See also* Windows CE
  accessories, **36-38**
  bar code readers, **38**
  Handheld PCs (HPCs). *See* Handheld PCs (HPCs)
  other, **34–35**, *35*
  overview of types, **12**
  Palm-size PCs (PPCs). *See* Palm-size PCs (PPCs)
  tablet devices, **35**, *35*
CeCheckPassword(), **476–477**
CeClearUserNotification(), **477**
CeCloseHandle(), 289, **477**
CeCopyFile(), **477–478**
CeCreateDatabase(), 122, 292, **478–479**
CeCreateDirectory(), **479**
CeCreateFile(), 289, **479–480**
CeCreateProcess(), 292, **480–481**
CEDB_AUTOINCREMENT flag, **296**
CEDB_sort types, **124**
CeDeleteDatabase(), 180, 292, **481–482**
CeDeleteFile(), **482**
CeDeleteRecord(), 140, 292, **482**
CeFindAllDatabases(), **482–483**
CeFindAllFiles(), **484**
CeFindClose(), 288, **484–485**
CE_FIND_DATA, **289**
CeFindFirstDatabase(), **292**, 293
CeFindFirstFile(), **288**
CeFindNextDatabase(), 176, 292, **486**
CeFindNextFile(), 288, **486–487**
CeGetClassName(), 293, **487**
CeGetDesktopDeviceCaps(), 285, 286, *286*, **487–488**
CeGetDeviceId(), **488**
CeGetFileAttributes(), **489**
CeGetFileSize(), **489**
CeGetFileTime(), **490**
CeGetLastError(), **490–491**
CeGetSelectedDeviceId(), **491**
CeGetSpecialFolderPath(), 289, 291, **491**
CeGetStoreInformation(), **492**
CeGetSystemInfo(), 285, **492**
CeGetSystemMetrics(), **493**
CeGetSystemPowerStatusEx(), 285, **493–494**
CeGetTempPath(), 289, **494**
CeGetUserNotificationPreferences(), **494–495**
CeGetVersionEx(), 288, **495–496**
CeGetVersionInfo(), **285**
CeGetWindow(), 292, **496**
CeGetWindowText(), 292, **497**
CeGlobalMemoryStatus(), 285, 286, **497**
CeHandleAppNotifications(), **498**
CeMoveFile(), **498**
CEOID (Ce Object Identifier), **125**, 127, 132
CeOIDGetEx(), **285**
CeOidGetInfo(), 180, **498–499**
CeOpenDatabase(), 128, 137, 180, 181, 292, **499**
CEPROVAL structures, **137–139**, 176–177
CeRapiFreeBuffer(), **500**
CeRapiGetError(), **500**
CeRapiInit(), 283–284, 293, **500–501**
CeRapiInitEx(), **501**
CeRapiInvoke(), **501–502**
CeRapiUninit(), **283–284**, **502**
CeReadFile(), 289, **502–503**
CeReadRecordProps(), 136, 138, 292, **503–504**
CeRegCloseKey(), 287, **504**
CeRegCreateKeyEx(), 287, **504–505**

CeRegDeleteKey(), 287, **505**
CeRegDeleteValue(), 287, **506**
CeRegEnumKeyEx(), 287, **506–507**
CeRegEnumValue(), 287, **507–508**
CeRegOpenKeyEx(), 287, **508–509**
CeRegQueryInfoKey(), 287, **509–510**
CeRegQueryValueEx(), 287, **510–511**
CeRegSetValueEx(), 287, **511–512**
CeRemoveDirectory(), **512**
CeRunAppAtEvent(), **512–513**
CeRunAppAtTime(), **513**
CeSeekDatabase(), 131, 132, 133, 134, 135, 180, 181, 292, **513–514**
CeSetDatabaseInfo(), 182, 292, **514–515**
CeSetEndOfFile(), **515**
CeSetFileAttributes(), **515–516**
CeSetFilePointer(), **516**
CeSetFileTime(), **516–517**
CeSetUserNotification(), **517–518**
CeSHCreateShortcut(), 292, **518–519**
CeSHGetShortcutTarget(), 292, **519**
CEVT_BLOB type, **297**
CEVT_field types, **123–124**
CeWriteFile(), 289, **519–520**
CeWriteRecordProps(), 139–140, 292, **520–521**
CFrameWnd, **192**, **193**
char types, **43–46**
checking return result of memory allocation, **57**
chips
  CE device type, **9**
  compilers and, **147–148**
_chsign(), **438**
_clearfp(), **439**
CListView control. *See* ListView controls
closing
  applications, **166–168**
  AreaCodeDatabaseFinish(), **140**
  close (X) button, 25, 204, **418**
  CloseAddressBook(), **521**
  CloseDB(), **185**
  CloseHandle, **88–89**
  debuggers, **229**
  fclose(), **88–89**
  Logo program applications, **418**
  Task Manager, **167**
CMainFrame
  InsertButtons(), **204–206**
  OnCreate() event, **203**
CObArray, **179**
codebase defined, **68**
color
  Handheld PC/Pro devices (HPC/Pros), **29**
  Handheld PCs (HPCs), **26**
  support for, **430**
  Task Manager, **169**
ComboBox, **270–271**, *271*
command bars, **415**, 416, 419
command line options, **344**
CommandBands_AddAdornments(), **521–522**
CommandBands_AddBands(), **522–523**
CommandBands_Create(), **523–524**
CommandBands_GetCommandBar(), **524**
CommandBands_GetRestoreInformation(), **525**
CommandBands_Height(), **525**
CommandBands_IsInvisible(), **526**
CommandBands_Show(), **526**
CommandBar_AddAdornments(), **526–527**
CommandBar_AddButtons(), 527–528
CommandBar_AddToolTips(), **528**
CommandBar_Create, **528–529**
CommandBar_DrawMenuBar(), **529**
CommandBar_GetMenu(), **529**
CommandBar_Height(), **529–530**
CommandBar_InsertButton(), **530**
CommandBar_InsertComboBox(), **530–531**
CommandBar_InsertMenuBar(), **531**
CommandBar_InsertMenubarEx(), **531–532**
CommandBar_IsVisible(), **532**
CommandBar_Show(), **532**
commandments for writing for Windows CE. *See* writing for Windows CE
Common Dialog, **234–235**, *235*
communications from CE to outside world. *See* CE communications
Compact Flash (CF) cards
  detecting, **319**
  Ethernet, **37**
  supporting installation to, **414–415**
compiling
  chips, **147–148**
  compiler defines, **68–69**, *69*
  conditional defines, **69–70**
  generic string types, **44–45**
  MFC applications for Windows CE, **200**
  runtime platform detectors, 69, **70–72**
  text literals, **46**
  toolkit for VC++, **146–147**

compressing data, **357**
conditional defines, **69–70**
configuring applications, **344**
connecting to CE devices, **150**
controls
  ActiveX Control Pack, **231–235**, *232*, *235*
  Control Manager
    grid control, **208**
    VBCE, **230–231**
  _controlfp(), **439**
  Image Controls, **250**, 268–269
  standard VBCE controls, **235**, *235–236*
  unsupported standard VB controls, **238**
converting
  MFC applications to CE
    Carcalc, **221–222**, *221*
    mechanical issues. *See* MFC applications ported to CE
  VB applications to CE
    array declarations, **256–260**, *257*
    ATM conversion overview, **244–249**, *245*, *246*, *276*, *276*
    Bitmaps, **250**
    change Global to Public, **249**
    country and bit maps, **266–269**
    Declare statements, **246–247**
    eliminating code, **274–275**
    eliminating forms, **270–274**, *271*
    eliminating optional controls and DLLs, **274**
    Explicit Type declarations, **249**
    Explicit Types, **249**
    icons, **250**
    Image Control, **250**
    mechanical issues, **249–261**, *257*
    modal forms, 248, **253–255**
    modifying resource loads, **255–260**, *257*
    mouse/cursor code, **252–253**
    optimizing, **270–275**
    optional features, **262–269**, *264*, *265*
    planning porting process, **247–249**
    porting from VB to VBCE, **247**
    preparing frmAmountWithdrawn Form, **261**
    sound, **263–265**, *264*, *265*
    Typecasts, **250**
    using Developer Studio, **256–257**, *257*
    WAV files, **250–253**, 262–266, *264*, *265*
  Unicode- to ANSI-based text files, **49–51**, *49*, *51*
copying
  comparison between third-party database engines, **357**
  _copysign(), **439**
  file copy in database administration, **343**
cos(), **451**
cosh(), **452**
cost analysis
  application testing, **408**
  development machine choices, **372**
  HPC/Pros, 369, *369*, 372
  HPCs, 365, 372
  laptops, **368**, *369*
  mobile devices, **348**
  Oracle Lite, **347**
  PPCs, 364, 365, 372
country bitmaps, **266–269**
CPrintDialog, **194**
CPUs
  determining type over Internet, **413**
  Logo program requirements, **427**
creating
  Cab archive, **397–398**
  Cab Wizard setup programs, **399–402**
  CE database, **122–127**
  conditional defines, **69–70**
  CreateAddressBook(), **533**
  CreateDB(), **183–184**
  CreateDC(), **30–31**
  CreateFile(), **80–81**
  CreateFileForMapping(), **533–534**
  CreateHeap(), **60**
  CreateProcess(), **81**
  custom<Meta>Tag, **390**
  INF files, **392–397**
  INI files, **398–399**, *398*
  multiple-file help system files, **385–391**, *389*
  Project Wizards for dialog-based applications, **154–157**, *155*, *157*
  runtime platform detectors, **70–72**
  setup programs, **399–402**
  single-file help system files, **380–385**
CShoppingListDoc, **182–183**
CShoppingListView
  methods of, **187**
  OnDelete(), **189–190**
  UpdateListBox(), **187–189**

Ctrl+letter, **425**
currency conversions, **246**
cursors, 420, 422–423
custom <meta> tags, **389–391**
Custom App Wizard dialog, *157*
Customer List report, **345–346**, *345*
Cwnd, **192**, **193**

D

databases
  access functions, **292**
  engines. *See* Windows CE Database Engine
  schema, **342–343**
dbcd_devicetype, **322–323**
DBView, **63–66**, *63*, *64*, *66*, *67*
debugging
  closing debugger, **229**
  Ethernet, 36, **151**
  generic SDK tools, **158**
  HTML-based help, **378**
  TCP/IP Transport, **150–151**
  VB IDE, **228**
  VBCE, **228–229**, *229*
  Windows CE services, **150**
declaring
  Unicode strings, **43–45**, *44*, *45*
  VB alternatives to, **246–247**, 251
defaults
  database administrator userid, **344**
  drag multi-select, **427**
  ListView controls, **427**
  specifying default destination folders, **402**, *402*
  system fonts, **423**
#define, **69–70**
deleting
  AreaCodeDatabaseDelete(), **125–126**, 140
  CCeDBDatabase methods for, **180**
  CeDeleteDatabase(), 180, **481–482**
  CeDeleteFile(), **482**
  CeDeleteRecord(), 140, **482**
  data in ListView control, **212–217**, *214*, *217*
  DDLS, **274**
  DeleteAddressCard(), **534**
  DeleteAndRenameFile(), **534–535**
  forms from VB applications converting to CE, **270–274**, *271*
  indexes, **130**
  OnDelete(), **189–190**
  optional controls, **274**
  using EditAdd Record, **136**, *136*
Delphi
  Cab Wizard setup programs, **399–402**
  properties viewer, **299–302**, *299*
demo, **344–347**, *345*
DeregisterDevice(), **535**
Developer Studio, *257*
development machine choices
  conclusion, **372**
  cost analysis, **372**
  HPC devices, **371**
  HPC/Pro devices, **371**
  overview, **370**
  PPC devices, **370–371**
Device Property dialog, **150**
dhrystone ratings, **10–11**
dialog-based applications, **154–157**, *155*, *157*
difftime(), **452**
digital cameras, **367**, *369*
disk file replication, **354**
displays
  display size CE devices, **9**
  display type CE devices, **9**
  Handheld PC/Pro devices (HCP/Pros), **29**
  Handheld PCs (HPCs), **26**
  Palm-size PCs (PPCs), **12–15**, *14*, *15*
  user data in ListView control, **210–212**
distributing CE applications
  help files
    creating, **378**
    multiple-file system, **388–391**
    single-file system, **380–385**, *381*
    types of, **379–380**
  installing programs on user's CE device
    Cab Wizard option, **392–402**, *398*
    InstallShield, **402–405**, *402*, *403*, *404*, *405*
    overview, **391–392**
  overview, **378**
div(), **452**
DLLs
  converting WAV files, **250**
  deleting, **274**
  driver names, **30**
  RAPI as, **283**
  storage space, **161**
  VGA-out port, **29–30**

dwSeekType, **131**
dynamic vs. static linking, **299**

E

editing
  capitalization in edit boxes, **430**
  Edit/Add Record dialog, **135–135**, *135*, *136*
  grid data, **232–233**
embedded applications with VB, **227–228**
emulation
  emulators for platform SDKs, **158–160**
  WAV files, **265**
  _WIN32_WCE_EMULATION, **68–69**
EnableEUDC(), **535–536**
EnableHardwareKeyboard(), **536**
Enterprise Manager applications, **122**
entry tags, **387–388**
EnumPnpIds(), **318**
Ethernet
  debugging, 36, **151**
  networking solutions
    Compact Flash (CF) Ethernet cards, **37**
    overview, **36**
    PCMCIA-based "Low Power" Ethernet cards, **37**
    Proxim Range LAN2, **37**, 151, 366
    Socket Communications, **37**
    wired connections, **37**
_evct(), **439**
exception handling methods, **79–80**
EXE file size, **56–57**, 161
Exists(), **184**
Exiting
  Logo program requirements, **429**
  Palm-size PCs (PPCs), **24–25**
exp(), **452**
Explicit Type declarations change, **249**
Explicit Types change, **250**

F

fabs(), **453**
fclose(), **88–89**
_fcvt(), **439**
FEFF signature, **48**
fgetc(), **89–90**
fgets(), **92–94**
fields
  not table-specific, **120–121**
  sorting based on one, **129**
files
  FILE*-based functions, **81–83**
  file access functions, **288–290**
  file handles defined, **81–82**
  file pointers defined, **81**
  file-based replication, Oracle Lite for CE, **354**
  Files dialog, **403–404**, *403*, *404*
  Logo program file-handling requirements, **431–433**
  RAPI file access functions, **288–290**
finding
  FindFirstEntry(), **536–539**
  previous instance, **418**
  substitute functions using C/C++ language, **77–79**
_finite(), **440**
flag bitmaps, **266–269**
floats, **185**
floor(), **453**
fmod(), **453**
form factors
  compiler defines, **68–69**, *69*
  creating conditional defines, **69–70**
  creating runtime platform detectors, **70–72**
  DBView, **63–66**, *63*, *64*, *66*, *67*
  overview, **61–62**
  single codebases, **67–69**
  tailoring application UI to device, **62–67**, *63*, *64*, *66*, *67*
format string pointer, **100**. *See also* LPWSTR
_fpclass(), **440**
fpieee_flt(), **440**
_fpreset(), **440**
fprint(), **97–99**
fputc(), **90–91**
fputs(), **94–95**
free(), **453**
FreeAddressCard(), **539–540**
frmAmountWithdrawn, **261**
frmInput, **270–273**
frmOpen, **270–274**

fscanf(), **99–107**
fseek(), **108–109**
FtpCommand(), **540–541**
fwrite(), **96–97**

G

_gcvt(), **441**
generic
  dialog base for Project Wizard, **154–157**, *155*, *157*
  platform SDKs, **158**
  string types, **44–45**, *45*
GetAddressCardIndex(), **541**
GetAddressCardOid(), **542**
GetAddressCardProperties(), **542**
GetClipboardDataAlloc(), **543**
GetColumnProperties(), **543**
GetDesktopWindow(), **164–165**
GetMatchingEntry, **544**
GetMessageSource(), **544–545**
GetMouseMovePoints(), **545**
GetNumberOfAddressCards(), **546**
GetPropertyDataStruct(), **546**
GetSortOrder(), **547**
GetStoreInformation(), **547**
GetSystemMetrics, **70**, *72*
GetSystemPowerStatusEx(), **548**
GetSystemTime(), **159–160**
GetType(), **178**
GetWindowText(), **53**
Global to Public change, **249**
Grids
  MFC applications ported to CE, **207–217**, *214*, *217*
  runtime and, *232*
  solving read-only problem, **232–233**
  third-party MFC library, **199**
  VCBE, **231–233**, *232*

H

Handheld PC/Pro devices (HPC/Pros). *See also* Hewlett-Packard Jornada HPC/Pro devices
  application templates, **226–227**
  CapEdit control, **34**
  choosing development machine, **371**
  choosing machine for development, **371**
  cost analysis, 369, *369*, 372
  displays, **29**
  HTML Viewer controls, **34**
  Ink control, **34**
  insurance field agents case study, **368**, 369, *369*
Handheld PCs (HPCs)
  Alt-tap behavior, **78**
  application templates, **226–227**
  CapEdit control, **27**
  choosing machine for development, **371**
  cost analysis, **365**, *365*, 372
  DBView, **63–67**, *63*, *64*
  displays, **26**, **29**
  form factor choices, **364**
  Handheld PC/Pro devices (HPC/Pros)
    overview, **27–28**, *28*
    version numbering of operating system, **34**
    VGA-out ports, **29–33**
  HTML Viewer control, 25, **27**
  Ink control, **27**
  Internet document retrieval, **27**
  keyboards, **26**
  Logo program
    functionality requirements, **427–431**
    installation requirements, **411–415**
    UI requirements, **415–427**
  overview, **25**, *26*
  predefined destination directories, **394**
  tailoring application UI to, **62–67**, *62*, *63*, *66*
  version numbering of operating system, **34**
HANDLE, **82**, 84, 86
handwritten data, **23**
hardware aspects of CE communications
  IR ports, **307**
  modems, **307**
  overview, **306**
  PCMICIA cards, **307**
  serial ports, **307**
HBRUSH, **31–33**
headers, **386**
HeapAlloc(), **59–60**
"Hello World" application, *153*
Help "?" button, **416**
help files

creating, **378–379**
help viewer, **391**
HTML-based, 378, **426**
Logo program testing, **428**
multiple-file system
FrontPage, **390**
header, **386**
launching help viewer, **391**
overview, **385–386**, *386*
table of contents, **387–388**
pop-up, **425–426**
single-file system
<!—PegHelp—> comment, **383**
additional HTP file tricks, **384–385**
FrontPage, **382**
headers, **382**
HTP files, **382**
overview, **380–382**, *381*
Hewlett-Packard Jornada HPC/Pro devices
Alt-click combination, **8**
application templates, **227**
mouse, **7**
overview, **27–28**, *28*
hierarchy in registry, **115**, *115*
HKEY_CLASSES_ROOT, **115**
HKEY_CURRENT_USER, **115**
HKEY_LOCAL_MACHINE, **115**, 118, 314, 319
HKEY_USER, **115**
hourglass or wait cursor, **420**
HPC/Pros. See Handheld PC/Pro devices (HPC/Pros)
HpcLadr, **338–341**, *339*
HPCs. *See* Handheld PCs (HPCs)
HTC files
header, **386**
overview, **385–386**, *386*
table of contents, **387**
table of contents entry tags, **387–388**
HTML
help, **426**
InitHTMLControl(), **550**
Viewer controls
Handheld PC/Pro devices (HPC/Pros), **34**
Handheld PCs (HPCs), 25, **27**
HTP files
<—PegHelp—> comment, **383**
additional tricks, **384–385**
custom<Meta>Tag, **390**
in multiple-file help system, **388–389**, *389*
overview, **380–382**, *381*
_hypot(), **441**

I

icons, **250**, 425
IDC_LISTTASKS, **163–168**
Image Controls, **250**, 268–269
ImageList
ImageList_Duplicate(), **548–549**
VBCE, **234**
improper uses of Windows CE registry, **116–119**
Include paths, **147–148**
indexing
creating sort order at table creation, **130**
deleting indexes, **130**
four sort order maximum, **129–130**, *130*
Modify Indexes dialog box, **66**, *66*, *67*
only one active sort order per database, **129–130**
overview, 114, **122–129**
sorting based on one field, **129**
INF files, **392–397**
Infrared Sockets (IrSock), **314**, 332
INI files, **398–399**, *398*
InitCommonControlsEx(), **549–550**
InitHTMLControl(), **550**
InitInkX(), **550**
Ink control
Handheld PC/Pro devices (HPC/Pros), **34**
Handheld PCs (HPCs), **27**
Palm-size PCs (PPCs), **18–22**, *19*
Input Panels
Logo program requirements, 424, 430
Owner applet, **19–22**, *19*
Palm-size PCs (PPCs), 15, 16–18, **18–22**, *19*, 20–22, 79
raising and lowering, **16–18**, *16*
scroll bars, 20–22
inserting
AreaCodeDatabaseInsert(), **139–140**
InsertButtons(), **204–206**
mechanical issues porting MFC applications to CE, **204–206**

installing
InstallDir, **393**, *396*
Logo program requirements, **411–415**
programs to user's CE device
Cab Wizard, **392–402**, *398*
InstallShield, **402–405**, *402*, *403*, *404*, *405*
insurance field agents case study
conclusion, **370**
cost analysis, **368–369**, *369*
overview, **366–367**
technical issues
digital cameras, **367**
HPC/Pro, **368**
overview, **367**
InterlockedTestExchange(), **551**
Internet
determining user's CPU type over, **413**
document retrieval, **27**
Logo program installation, 411, **413**
Oracle Lite for CE, **353–354**
replication, **353–354**
Inventory Management System
case study
bar code scanner, **363**
conclusion, **366**
cost analysis, **365–366**, *365*, *366*
form factor choices, **364**
Handheld PC (HPC), **364**
Palm-size PC (PPC), **364**
synchronization vs. wireless connection, **363–364**
technical issues, **362–364**
overview, **362**
invoicing systems, **337–338**
IR communications
Infrared Sockets (IrSock), **314**, 332
IR ports, **307**, 420
IR transfer of files, **433**
IrCOMM, **312–314**, *313*
IrDA standards, **420**
IrDetect application, *313*
overview, **309**
Raw IR, **309–312**
IrDetect application, *313*
_isnan(), **441**
iswalnum(), **454**
iswalpha(), **454**
iswascii(), **454**
iswcntrl(), **454**
iswctype(), **455**
iswgraph(), **455**
iswlower(), **455**
iswprint(), **455**
iswpunct(), **456**
iswspace(), **456**
iswupper(), **456**
iswxdigit(), **456**
_itoa(), **441**
_itot(), **310**
_itow(), 310, **441**

J

_j0(), **442**
_j1(), **442**
_jn(), **442**

K

keyboards
EnableHardwareKeyboard(), **536**
Handheld PCs (HPCs), **26**
hints for HPC applications, **419**
Palm-size PCs (PPCs) lacking, **15**
toolbar button equivalents, **425**
keys defined, **115**

L

labs(), **457**
ladder puzzles, **338**
language selection, **245**, 274–275
laptops
case study of use of, **368–369**, *369*
vs. HPC/Pros devices, **366**
HPC/Pros keyboard size comparison, **34**
Sony Vaio, 368
ldexp(), **457**
ldiv(), **457**
LIB paths, **147**
license agreement, **341**

linking, **299**
ListView controls
  database administration, **346**
  deleting data, **212–217**, *214*, *217*
  displaying user data, **210–212**
  drag multi-select, **427**
  grid replacements, **208–209**
  managing user data while program runs, **209**
  as replacement control, **208–209**
  Report View style, **129**, *130*
  user entry, **212–217**, *214*, *217*
  VBCE, **234**
LMEM_ZEROINIT, **77**
LoadResString, **260**
LocalAlloc(), **58–60**, *77*, *84*
log(), **457**
log10(), **458**
_logb(), **442**
Login dialog, **344–345**, *345*
Logo program
  file handling requirements
    common dialog boxes, **431**
    e-mailing files, **433**
    file system display, **432**
    IR file transfers, **433**
    MRU lists, **433**
    multi-instance applications, **432**
    open document name display, **432**
    shortcut to recently used documents, **432–433**
    taskbar buttons, **432**
  functionality requirements
    applications storing and loading their state, **430**
    capitalization in edit boxes, **430**
    CE-supported CPUS, **427**
    color support, **430**
    dialogs and Input Panel, **430**
    exiting without user intervention, **429**
    graphical components, **428**
    help files, **428**
    long names in file-related operations, **431**
    RAPI, **429**
    testing, **428**
    windows, **428**
    Winsock, **429**
    WM_HIBERNATE messages, **429**
installation requirements
  Application Manager, **411**
  Cab files, **411**, 412
  compact flash cards, **414–415**
  copying files to subdirectory of CEAppManager Path, **412**
  installing from desktop to device, **412**
  installing from Internet to device, **413**
  naming program files, **414**
  nonshared files, **411**
  removing user settings from CE registry, **412**
  shared program files, **412**
  shortcut on Start menus, **415**
  uninstalling, **412**, 414
overview, **408–410**
UI requirements
  Address Book API, **419–420**
  Alt+Enter, **422**
  Alt+H for Help, **416**
  Alt+letter, **425**
  Alt+tap, **422**
  buttons, **416**, 417, 419, 424, 425
  centering dialogs, **423**
  closing, **418**, 426–427
  command bars, **415**, 416, 419
  Ctrl+letter, **425**
  default system font, **423**
  desktop as parent window, **425**
  drag multi-select, **427**
  finding previous instance, **418**
  games, **415**
  Help "?" buttons, **416**
  hints for HPC Applications, **419**
  hints for PPC Applications, **419**
  hourglass or wait cursor, **420**
  HTML-based help, **426**
  icons for files types, **419**
  IR port, **420**
  IrDA standards, **420**
  keyboard equivalents for toolbar buttons, **425**
  keyboard shortcuts, **419**
  launching, **426–427**
  ListView controls, **427**
  menus, **416**, 417, 422, 423
  mouse cursors, **422–423**
  msgstore API, **420**

order of standard functions, **417**
pop-up help, **425**
pop-up menus, **422**, 423
property sheets, **422**
rebars, **415**
resizing, **421–422**
shortcut keys, **425**
showing/hiding Input Panel, **424**
size of dialogs, **424**
Start menu shortcuts, **425**
taskbar settings, **421–422**
title bar, **426**
tool tips, **419**
tray icons, **425**
underlined accelerators, **425**
windows, **426**
WM_RBUTTONUP, **423**
longjmp(), **458**
lowering
System Input Panels
codes provided for, **16–18**, *16*
Owner applet, **19**, *19*
low-memory environment
checking return result of memory allocation, **56–57**
different types of memory allocation, **59–60**
EXE file size, **56–57**
handling WM_HIBERNATE message, **60–61**
mass-allocating application memory, **58–60**
minimizing static variables, **56**
overview, **55–56**
reactivating application, **61**
LPCTSTR, **45**
LPTSTR, **45**
LPWSTR, **44**, 84, 92–93, 94, 97
_lrotl(), **442**
_lrotr(), **443**
_ltoa(), **443**
_ltow(), **443**

M

macros
generic string types, **44–45**, *45*
macro strings, **394**
TEXT, **46**
MailClose(), **551**
MailDelete(), **551–552**
MailDeleteAttachment(), **552–553**
MailErrorMsg(), **553–554**
MailFirst(), **554–555**
MailFree(), **555**
MailGet(), **555–556**
MailGetAttachment(), **556**
MailGetField(), **557**
MailGetFolderId(), **557–558**
MailGetFolderName(), **558**
MailGetSort(), **558–559**
MailGetSvcId(), **559**
MailHeaderLen(), **560**
MailLocalAttachmentLen(), **560**
MailNext(), **560**
MailOpen(), **561**
MailOpenNotify(), **561**
MailPut(), **562**
MailPutAttachment(), **562–563**
MailPutFolder(), **563**
MailRequestAttachment(), **563–564**
MailSetField(), **564–565**
MailSetSort(), **565**
MailUpdate(), **566**
malloc(), **458**
mass-allocating application memory, **58–60**
_matherr(), **443**
MAX_PORTS, **310**
mbstowcs(), **458**
_memccpy(), **444**
memchr(), **459**
memcmp(), **459**
memcpy(), **459**
_memicmp(), **444**
memmove(), **459**
memory
allocating. *See* allocating memory
constraints in Windows CE, **5–7**
low-memory environment. *See* low-memory environment
WM_HIBERNATE messages, **429**
memset(), **460**
menus
Logo program UI requirements, **416**, 417, 422, 423
pop-up, **422**, 423

Save As ANSI menu, **51**, *51*
Start menu shortcuts, **425**
MFC applications
missing classes, **194**
modified classes, **191–193**
modified classes that gained functionality
CFrameWnd, **193**
CWnd, **193**
overview, **192–193**
modified classes that lost functionality
CDC, **192**
CFrameWnd, **192**
CWnd, **192**
overview, **192**
MoveToEx()/LineTo(), **78**
new classes for CE
CCeDBDatabase, 174, **179–182**, *180–182*
CCeDBEnum, 174, **176**
CCeDBProp, 174, **176–178**, *177*
CCeDBRecord, 174, **179**, *179*
CCeSocket, 174, **175–176**, *175*
overview, **174**
Palm-size PCs (PPCs), **117**
porting to CE. *See* MFC applications ported to CE
vs. SDK-style coding, **160–162**
using CCeDB classes
AddRecord(), **185–186**
CloseDB(), **185**
CreateDB(), **183–184**
database as document, **182–186**
database as View, **186–191**
OnDelete(), **189–190**
OnInsert(), **190–191**
OpenDB(), **184–185**
UpdateListbox(), **187–189**
MFC applications ported to CE
Carcalc, **221–222**, *221*
mechanical issues
ActiveX Grid control, **208**
grid control, **207–217**, *214*, *217*
InsertButtons(), **204–206**
ListView control, **208–217**
MFC 2.0 vs. MFC 2.1, **202**
printing support, **206–207**
status bars, **201–206**
toolbars, **201–206**
OLE Automation, **220–222**, *221*
optimizing for CE, **217–220**
overview, **198**
Shopping List application, **198–200**, *198*
minimizing, **56–57**
mobile devices, **348**. *See also* Oracle Mobile Agents
modal forms, 248, **253–255**
modems
built-in, **314–318**, *318*
classes of, **314**
hardware aspects, **307**
Modem Selector dialog, **318**
PCMCIA, **318**
standard external, **314**
modf(), **460**
Modify Indexes dialog box, **66**, *66*, *67*
ModifyAddressCard(), **566–568**
mouse
converting VB codes for, **252–253**
Hewlett-Packard Jornada HPC/Pro device, 7
Logo program support for mouse cursors, **422–423**
MoveToEx()/Line To(), **77–78**
MRU lists, **433**
msgstore, API, **420**
_msize(), **444**
multi-instance applications, **432**
multiple-file help system
header, **386**
HTP files in
custom <meta> tags, **389–391**
overview, **388–389**, *389*
launching help viewer, **391**
overview, **385–386**, *386*
table of contents, **387–388**

N

networking solutions. *See* Ethernet
_nextafter(), **444–445**
nonshared program files, **411**
notepad, **48–55**, *49*, *51*

O

ODBC support, **349**
Odyssey Software, **207**
OLE Automation, **220–222**, *221*
OnDelete(), **189–190**
opening
  CE database, **127–128**
  fopen(), **83–88**
  OpenAddressBook(), **568**
  OpenAddressCard(), **568–569**
  OpenDB(), **184–185**
  Unicode- and ANSI-based text files, **49–51**
optimizing
  MFC applications ported to CE, **217–220**
  VB code for CE, **274–275**
Oracle Lite for CE
  application development support, **349–350**
  copying data to desktop, **357**
  data compression, **357**
  disk file replication, **354**
  file-based replication, **254**
  Internet replication, **353–354**
  key features, **349**
  ODBC support, **349**
  overview, **347–349**
  replication, **352–354**
  sample applications, **350–352**, **356**
  security, **254**
  SQL support, **357**
  storage requirements, **357**
  stored procedures, **357**
  synching data with desktop, **357**
  user privileges, **358**
  wireless replication, **353**, *355*
Oracle Mobile Agents
  anytime, anywhere, **354**
  developing applications and agents, **355**
  optimized for mobile environment, **354–355**
  overview, **354**
  reliability, **355**
  security, **355**
  wireless replication, **355**
Owner applet, **19–22**, *19*

P

Palm-size PCs (PPCs)
  CapEdit control, **23–24**
  choosing machine for development, **370–371**
  cost analysis, **364–365**, *365*, *372*
  display, **12–15**, *14*, *15*
  exiting, **24–25**
  form factor choices, **364**
  Ink control, **22–23**
  keyboard vs. Input Panel, *15*, *16–17*, **18–22**, *19*, *20–22*, *79*
  Logo program
    file-handling requirements, **431–433**
    functionality requirements, **427–431**
    installation requirements, **411–415**
    UI requirements, **415–427**
  MFC applications, **117**
  overview, **12**, *13*
  physical memory, **6**
  predefined destination directories, **394**
  right-clicking, **8**
  System Input Panels
    detecting when user pops up, **18–22**, *19*
    no Alt key, **79**
    overview, **15**
    raising and lowering, **16–18**, *16*
    scroll bars, **20–22**
  tailoring application UI to, **62–67**, *62*, *64*, *67*
  Task Manager, **162–163**, *162*
  Taskbar, **24–25**
  version numbering of operating system, **34**
  voice recording and playback, **24–25**
  WM_RBUTTONDOWN message, **79**
passwords, *117*, *344*
PC/PCMCIA cards
  hardware aspects, **307**
  "Low Power" Ethernet cards, **37**
  software aspects
    PCMCIA slot overview, **318**
    when application starts, **318–320**, *320*
    when card changes, **321–326**, *323*
PDAs, **347–348**
Pentax Technologies, **207**
PIM (Personal Information Manager) applications, **419–420**
Platform Manager, **148–150**, *149*, *150*

platform SDKs
emulators, **158–160**
generic, **158**
overview, **158**
plug and play, **318**
PnpIds, **318–320**, *320*
PocketJet II printer, **207**
poid parameters, **127**
Poly-Line(), **78**
porting. *See* converting
pow(), **460**
predefined destination directories, **393–394**
Preprocessor defines, **70**
printing
CPrintDialog, **194**
MFC applications ported to CE, **206–207**
PocketJet II, **207**
proc_name, **343**
Project Wizards
for dialog-based applications, **154–157**, *155*, *157*
overview, **151–152**, *152*
WCE Application Wizard, **157**
WCE MFC AppWizard, **152–154**, *153*
proper uses of Windows CE registry, **116**
properties defined, **137**
property sheets, **422**
Proxim Range LAN2, **37**, 151, 366

Q

qsort(), **460**

R

raising
System Input Panels
Owner applet, **19**, *19*
writing codes for, **16–18**, *16*
rand(), **460–461**
Random(), **569**
RangeLAN2, **37**, 151, 365–366
RAPI (Remote API)
CeGetSpecialFolderPath(), **291**
database access functions, **292**
file access functions, **288–290**
general management functions, **283–284**
Logo program requirements, **429**
miscellaneous shell and system functions, **292–293**
overview, **282–283**
registry access functions, **287–288**
sample application, **293–298**, *294*
SDK tools and, **158**
system information functions, **285–286**, *286*
Unicode-based, **289**
using Delphi, **299–302**, *299*
RapiDBSave application, **293–294**, *294*
RapiFreeBuffer(), **569–570**
RasGetEntryDevConfig(), **570**
RasSetEntryDevConfig(), **570**
Raw IR, **309–312**
RDM/CE (Raima Data Manager for CE)
copying data to desktop, **357**
data compression, **357**
HpcLadr, **338–341**, *339*
license agreement, **341**
network models, **337–338**
relational data models, **337–338**
SQL support, **357**
storage requirements, **357**
stored procedures, **357**
synching data with desktop, **357**
user privileges, **358**
reactivating application, **61**
reading
AreaCodeDatabaseRead(), **136–139**
ReadFile, **80–81**, 90
read-only Grid control problem, **232–233**
realloc(), **461**
rebars, **415**
RecountCards(), **570**
recursive functions, **56**
RegisterDevice(), **570–571**
registry
access functions, **287–288**
application vs. information, **118**
BLOB (Binary Large Object) data, **118**
vs. database engine, **119**
hierarchy, **115**, *115*
improper uses of, **116–119**
MFC applications and Palm-size PCs (PPCs), **117**

proper uses of, **116**
redundant information, **118**
removing user settings from, **412**
storing system data, **116**
storing user-specific information, **116**
relational database systems, **120**, 337–338
remote functions
Remote File Viewer, **158**
Remote Heap Viewer, **158**, 159
Remote Process Viewer, **158**
Remote Registry Editor, **158**
Remote Spy++, **158**
Remote Zoomin, **158**, 159
replacement controls, **208–209**
replication
bi-directional, **356**
disk file, **354**
file-based, **354**
Internet, **353–354**
Oracle Lite for CE, **352–354**
Oracle Mobile Agents, **355**
wireless, **355**
reports
Report dialog, **345**, *345*
report_id, **343**
report_name, ASA (Adaptive Server Anywhere), **343**
REPSVR, **356**
resource loads, **255–260**, *257*
return result of memory allocation, **57**
_rotl(), **445**
_rotr(), **445**
RTL (runtime library) functions
choosing for Unicode strings, **47**, *47*
fgets(), **92**
fopen(), **83**
fprintf(), **97**
fputs(), **94**
fread(), **95**
fseek(), **108**
fwrite(), **96**
in Windows CE, **7**
runtime
grid control, *232*
library functions. *See* RTL (runtime library) functions
platform detectors, 69, **70–72**
TabStrip control, **233–234**, *233*
VBCE files, **229–230**

S

Sales Report, **346**
sample application
area code lookup utility, **135–140**
ASA (Sample Application Anywhere), **342–347**, *345*
ATM conversion, **244–276**, *245*, *246*, *276*
converting between Unicode and ANSI Character Sets, **48–55**, *49*, *51*
Oracle Lite for CE, **350–352**, 356
RAPI (Remote API), **293–298**, *294*
sample program in Odbc directory, **356**
Shopping List application, **198–200**, *198*
Stopwatch-like, **238–240**, *239*
Task Manager, **162–163**, *162*
Wizard for dialog-based applications, **154–157**, *157*
saving Unicode- and ANSI-based text files, **51–55**, *51*
_scalb(), **445**
scroll bars, **20–22**
SDK (Software Development Kit)
AutoPC SDK, **69**
platform, **158–160**
sample application
closing application, **166–168**
displaying task lists, **163–164**
keeping task list current, **168–170**
overview, **162–163**, *162*
replacing GetDesktopWindow(), **164–165**
switching to one task, **166**
using VC++, 146
searching
CCeDBDatabase seek methods, **181**
CeSeekDatabase(), 131, 132, 133, 134, 135, 180, 181, **513–514**
dwSeekType, **131**
given number of records, **134–135**
matching CEOID records, **132**
next record equal to search criteria, **134**
overview, **131–132**
records with equal value, **133**

records with greater values, **133**
records with smaller values, **132**
vs. seeking, **131**
using API for, **121**
security
ASA security_level, **343**
Oracle Lite for CE, **354**
Oracle Mobile Agents, **355**
serial communications, **308–309**
serial ports, **307**
SetColumnProperties(), **571**
SetDaylightTime(), **572**
SetFilePointer, **108–109**
SetMask(), **572–573**
SetSortOrder(), **573**
Setup programs, **399–402**
Setup Wizards
Cab Wizard, **399–402**
for VBCE, **236–237**
shared program files, **412**
SHCreateShortcut(), **573–574**
shell and system functions, **292–293**
SHGetShortcutTarget(), **574**
SHLoadDIBitmap(), **574**
Shopping List application, **198–200**, *198*
shortcuts
destination directories, **397**
displaying keys for, **425**
hints for HPC applications, **419**
Shortcut Properties dialog, **403**, *404*
Start menu, 425, 432
showing
Common Dialog control, **235**
SHShowOutOfMemory(), **575**
Sierra Imaging, **369**
sin(), **461**
single codebase, **67–69**
sinh(), **461**
size
dialogs, **424**
EXE files, **56–57**, 161
Handheld PC/Pro devices (HPC/Pros), **27**
Handheld PCs (HPCs), **25**, 62
Palm-size PCs (PPCs), 62
resizing based on taskbar settings, **421–422**
screen, 62, 70–72
string, **56**
of WAV files, **265–266**
windows resizing based on screen size, **428**
Socket Communications
bar code readers, **38**, 363
cost analysis, *365*
Ethernet debugging, **151**
serial I/O cards, **307**
wired network accessories, **37**
software aspects of CE communications
built-in modems, **314–317**, *318*
Infrared Sockets (IrSock), **314**, 332
IR communications, **308–314**, *313*
IrCOMM, **312–314**, *313*
modem-based communications, **314–318**, *318*
overview, **308**
Raw IR, **309–312**
serial communications, **308–309**
standard external modems, **314**
Sony Vaio, 368
sort orders. *See* indexing
SORTORDERSPEC, **123–127**
sound, 250–253, **262–266**, *264*, *265*
SQL (Structured Query Language)
alternate program's support for, **357**
CE database engine support for, **121**, 357
stored procedures, **342**
sqrt(), **461**
srand(), **462**
static variables, **56**
static vs. dynamic linking, **299**
status bars, **201–206**
_statusfp(), **445**
stdio.h file access functions, 52, 77
Stopwatch-like sample application, **238–240**, *239*
storage
misuse of CE registry, **116–117**
registry vs. database engines, **119**
structured. *See* structured storage
of system data, **116**
third-party database engines requirements, **357**
of user-specific information, **116**
VC++ (Visual C++), **161**
stored procedures, **342**, 357
strcat(), **462**
strchr(), **462**
strcmp(), **462**
strcpy(), **462**

strcspn(), **463**
strings
  allocating memory for (szChunk), **52–55**
  ANSI-based RTL functions, **47**, *47*
  ANSI-specific string type mappings
    generic to, **44–45**, *45*
    Unicode-specific to, **44**, *44*
  choosing RTL (runtime library) functions for Unicode, **47**, *47*
  declaring, **43–45**, *44*, *45*
  generic types, **44–45**, *45*
  macro, **394**
  size limits, **56**
  Unicode-specific types, **44**, *44*
  using Unicode strings for all text literals, **46**
  using Unicode types vs. char types, overview, **42–44**
  wide, **43**, *297*
strlen(), **463**
strncat(), **463**
strncmp(), **463**
strncpy, **463–464**
strstr(), **464**
strtok(), **464**
structured storage
  CE Database Engine API
    AreaCodeDatabaseDelete(), **140**
    AreaCodeDatabaseFinish(), **140**
    AreaCodeDatabaseInsert(), **139–140**
    AreaCodeDatabaseRead(), **136–139**
    overview, **135–136**, *135*, *136*
  indexing
    creating sort order at table creation, **130**
    four sort order maximum, **129–130**, *130*
    only one active sort order per database, **129–130**
    overview, **122–129**
    sorting based on one field, **129**
  overview, **114–115**
  registry
    application vs. information, **118**
    BLOB (Binary Large Object) data, **118**
    hierarchy, **115**, *115*
    improper uses of, **116–119**
    MFC applications and Palm-size PCs (PPCs), **117**
    proper uses of, **116**
    storing system data, **116**
    storing user-specific information, **116**
  searching
    given number of records from beginning of database, **134**
    given number of records from current record, **135**
    given number of records from end of database, **134**
    matching CEOID records, **132**
    next record equal to search criteria, **134**
    overview, **131–132**
    records with equal value, **133**
    records with greater values, **133**
    records with smaller values, **132**
  Windows CE database engine, vs. the registry, **118**
stubbing, **251–253**
stylus vs. mouse, **7–8**
substitute functions, **77–79**
Summary/confirmation dialog, **405**, *405*
_swab(), **446**
switch..case blocks, **178**
switch statements, **100**
swprintf(), **464**
swscan(), 104, **465**
swscanf(), **104**
Sybase's ASA (Adaptive Server Anywhere). *See* ASA (Adaptive Server Anywhere)
synching data
  Active Sync, **343**
  comparison between third-party database engines, **357**
  stored procedures, **342**
  vs. wireless connection, **363–364**
system information functions, **285–286**, *286*
System Input Panels
  detecting when user pops up, **18–22**, *19*
  lowering, **16–18**, *16*
  Owner applet, **19–22**, *19*
  raising, **16–18**, *16*
  scroll bars, **20–22**
SystemParametersInfo(), **70**
szChunk, **52–55**

T

table of contents, **387–388**
tables, **120–121**
tablet devices, **35**, *35*
TabStrip control, **233–234**, *233*
Tahoma (9-point), **423**
tan(), **465**
tanh(), **465**
task lists, displaying, **163–164**, 168–170
Task Manager, **162–163**, *162*, *167*
Taskbars
  changes while application running, **421–422**
  creating main window, **421**
  displaying open documents, **432**
  Logo program, **421–422**
  Palm-size PCs (PPCs), **24–25**
TCHAR, 45, 46, 57, **84**, 85–88, 90, 91, 94, 98, 100–107, 346
text literals, **46**
TEXT macros, **46**
Third-party database engines
  ASA (Adaptive Server Anywhere)
    administration and management, **343**
    command line options, **344**
    configuring application, **344**
    copying data to desktop, **357**
    Customer List report, **345–346**, *345*
    data compression, **357**
    database schema, **342–343**
    demo, **344–347**, *345*
    Login dialog, **344–345**, *345*
    overview, **341**
    proc_name, **343**
    Report dialog, **345**, *345*
    report_id, **343**
    report_name, **343**
    Sales Report, **346**
    sample application, **342–347**, *345*
    security_level, **343**
    SQL support, **357**
    storage space requirements, **357**
    stored procedures, **357**
    synching data with desktop, **357**
    user privileges, **358**
  comparison of features, **356–358**
Oracle Lite
  access to corporate data on mobile devices, **348**
  copying data to desktop, **357**
  costs, **347**
  data compression, **357**
  data-intensive applications on PDAs, **348**
  overview, **347**
  processing power deficiency, **347–348**
  SQL support, **357**
  storage requirements, **357**
  stored procedures, **357**
  synching data with desktop, **357**
  user privileges, **358**
Oracle Lite for CE
  application development support, **349–350**
  disk file replication, **354**
  file-based replication, **254**
  Internet replication, **353–354**
  key features, **349**
  ODBC support, **349**
  overview, **348–349**
  replication, **352–354**
  sample application, **350–352**, **356**
  security, **254**
  wireless replication, **353**
Oracle Mobile Agents
  anytime, anywhere, **354**
  developing applications and agents, **355**
  optimized for mobile environment, **354–355**
  overview, **354**
  reliability, **355**
  security, **255**
  wireless replication, **355**
overview, **336–337**
RDM/CE (Raima Data Manager for CE)
  copying data to desktop, **357**
  data compression, **357**
  HpcLadr, **338–341**, *339*
  license agreement, **341**
  network models, **337**
  overview, **337**
  SQL support, **357**
  storage requirements, **357**
  stored procedures, **357**
  synching data with desktop, **357**
  user privileges, **358**

title bars, **426**
tool tips, **419**
toolbars, **201–206**, *425*
toolkits
  for VB (Visual Basic language). *See* VB (Visual Basic language)
  for VC++. *See* Windows CE toolkit for VC++
towlower(), **465**
towupper(), **466**
TransparentImage(), **575–576**
tray icons, **425**
TreeView
  vs. DBView, **65**
  VBCE, **234**
try..catch, **79–80**
Typecasts change, **250**

U

UI
  Logo program requirements, **415–427**
  tailoring to device, **62–67**, *62*, *63*, *64*, *66*, *67*
_ultoa(), **446**
_ultow(), **446**
underlined accelerators, **425**
Unicode Character Set
  vs. ANSI-based functions, **97**
  applications must use, **42–43**
  converting between Unicode and ANSI
    opening text files, **49–51**
    overview, **48–49**, *49*
    saving text files, **51–55**, *51*
  declaring strings using Unicode vs. char types
    generic string types, **44–45**, *45*
    overview, **43–44**
    Unicode-specific string types, **44**, *44*
  FEFF signature, **48**
  file access, *95*
  fopen(), **84**
  RTL functions for Unicode strings, **47**, *47*
  string function equivalents, **47**, *47*
  using with other text files, **48**
  using Unicode strings for all text literals, **46**
  wstrlen(), **95**
UpdateListBox(), **187–189**
user privileges, **358**

V

variable argument lists, **97–107**
VarYYYYFromZZZZ() functions, **576–584**
VB (Visual Basic language)
  changes in, **237–238**, *238*
  converting real VB application to CE
    array declarations, **256–260**, *257*
    ATM conversion overview, **244–249**, *245*, *246*, *276*, *276*
    Bitmaps, **250**
    change Global to Public, **249**
    country and bit maps, **266–269**
    Declare statements, **246–247**
    eliminating code, **274–275**
    eliminating forms, **270–274**, *271*
    eliminating optional controls and DLLs, **274**
    Explicit Type declarations, **249**
    Explicit Types, **249**
    icons, **250**
    Image Control, **250**
    mechanical issues, **249–261**, *257*
    modal forms, 248, **253–255**
    modifying resource loads, **255–260**, *257*
    mouse/cursor code, **252–253**
    optimizing, **270–275**
    optional features, **262–269**, *264*, *265*
    planning porting process, **247–249**
    porting from VB to VBCE, **247**
    preparing frmAmountWithdrawn Form, **261**
    sound, **263–265**, *264*, *265*
    Typecasts, **250**
    using Developer Studio, **256–257**, *257*
    WAV files, **250–253**, 262–266, *264*, *265*
  standard controls, **235**, *235–236*
  Stopwatch-like application, **238–240**, *239*
  toolkit for CE
    ActiveX Control Pack, **231–235**, *235*
    application templates, **226–227**, *227*
    control manager, **230–231**
    debugger, **228–229**, *229*
    embedded applications, **227–228**
    overview, **226**
    runtime files, **229–230**
    Setup Wizard, **236–237**
    standard VBCE controls, **235**, *235–236*
    versions 5 and, 6, **238**

VBCE Miscellaneous Utility Control, **266**
VBScript, **237**
VC++ (Visual C++)
  using to develop for Windows CE, **146**
  Windows CE toolkit for
    compilers, **146–147**
    debugging, **150–151**
    Embedded Toolkit, **227–228**
    MFC vs. SDK-style coding, **160–162**
    overview, **146**
    Platform Manager, **148–150**, *149*, *150*
    platform SDKs, **158–160**
    Project Wizards, **151–157**, *152*, *153*, *155*, *157*
Veritest, **408**
version numbering of CE operating systems, **34**
VGA-out ports, Handheld PC/Pro devices (HPC/Pros), **29–33**
VirtualAlloc(), **60**
vswprintf(), **466**

W

WAV files, 250–253, **262–266**, *264*, *265*
WCE Application Wizard, **157**
WCE MFC AppWizard, **152–154**, *153*
WCHAR, **44**, 45
wcscat(), **466**
wcschr(), **467**
wcscmp(), **467**
wcscpy(), **467**
wcscspn(), **467**
_wcsdup(), **446**
_wcsicmp(), **447**
wcslen(), **468**
_wcslwr(), **447**
wcsncat(), **468**
wcsncmp(), **468**
wcsncpy(), **468–469**
_wcsnicmp(), **447**
_wcsnset(), **447–448**
wcspbrk(), **469**
wcsrchr(), **469**
_wcsrev(), **448**
_wcsset(), **448**
wcsspn(), **469–470**
wcsstr(), **470**
wcstod(), **470**
wcstok(), **470**
wcstol(), **471**
wcstombs(), **471**
wcstoul(), **471**
_wcsupr(), **448**
whitespace, **101**, 102, 103
whois program, **327–332**, *328*
wide strings, **43**, *297*
Windows CE
  database engine. *See* Windows CE Database Engine
  overview of. *See* Windows CE overview
  registry hierarchy. *See* registry
  toolkit for VC++. *See* Windows CE toolkit for VC++
  vs. Windows 98/NT, **5–9**
  writing for. *See* writing for Windows CE
Windows CE Database Engine
  API
    AreaCodeDatabaseDelete(), **140**
    AreaCodeDatabaseFinish(), **140**
    AreaCodeDatabaseInsert(), **139–140**
    AreaCodeDatabaseRead(), **136–139**
    overview, **135–136**, *135*, *136*
  vs. other familiar database engines
    databases as tables, **120**
    Enterprise Manager applications, **122**
    indexing, **122–130**
    management tools and utilities, **121–122**
    not relational, **120**
    organization and naming of data, **120**
    overview, **119**
    record-specific vs. table-specific fields, **120–121**
    searching, **131–135**
    similarities, **122–135**
    SQL (Structured Query Language) queries, **121**
  vs. the registry, **119**
Windows CE overview
  CE defined, **4–5**
  communications, **10**
  core OS features, **10**
  databases, **10–11**
  dhrystone ratings, **10–11**
  display size and type, **9**

hardware standardization, **8–9**
Internet features, **10**
low-memory situations, **6–7**
memory constraints, **5–7**
networking features, **10**
operating system requests, **6–7**
physical memory, **6**
power of, **11**
reduced API, **7**
reduced runtime library (RLT), **7**
resizing windows, **8**
right-clicking, **8**
running existing programs, **5**
stylus vs. mouse, **7–8**
unique features, **10–11**
user control over memory availability, **6–7**
vs. Windows 98/NT, **5–9**
Winsock 1.1, **10**
vs. x86 Intel platform, **9**
Windows CE toolkit for VC++
compilers, **146**
debugging, **150–151**
#define, **70**
GetDesktopWindow(), **164–165**
MFC vs. SDK-style coding
ease of development, **161–162**
overview, **160**
storage space, **161**
overview, **146**
Platform Manager, **148–150**, *149*, *150*
platform SDKs
emulators, **158–160**
generic tools, **158**
overview, **158**
Preprocessors defines, **70**
Project Wizards
for dialog-based applications, **154–157**, *155*, *157*
overview, **151–152**, *152*
WCE Application Wizard, **157**
WCE MFC AppWizard, **152–154**, *153*
sample SDK-style application
closing application, **166–168**
displaying tasks, **163–164**
keeping task lists current, **168–170**
overview, **162–163**, *162*
switching to one task, **166**
Windows NT, 159, 175, 192, 289, 326
Winsock 1.1
in CE communications, **327–332**
Infrared Sockets (IrSock), **314**, 332
Logo program requirements, **429**
overview, **10**
Wired network accessories for CE devices, **36–37**
wireless connections
accessories for CE devices, **36–37**
LAN Hub cost analysis, **366**
Oracle Mobile Agents, **355**
replication, **353**, 355
synching vs., **363–364**
Wizards
Cab Wizard, **392–402**, *398*
Project Wizards, 151–152, *152*, **154–157**, *155*, *157*
Setup Wizard for VBCE, **236–237**
WCE Application Wizard, **157**
WCE MFC AppWizard, **152–154**, *153*
WM_ACTIVATE, **61**
WM_COMMAND, **29**, 166, 167, 168
WM_DEVICECHANGE, **321–322**, 324, 326
WM_HIBERNATE, **60–61**, 193, 429
WM_INITDIALOG, **159–160**, 316
WM_RBUTTONDOWN, **78–79**
WM_RBUTTONUP, **423**
WM_SOCKET_NOTIFY, **176**
wrapper functions, **81**
WriteFile, **80–81**, 91
writing for Windows CE
first commandment of, **42–55**
form factors
compiler defines, **68–69**, *69*
creating conditional defines, **69–70**
creating runtime platform detectors, **70–72**
DBView, **63–66**, *63*, *64*, *66*, *67*
overview, **61–62**
single codebases, **67–69**
tailoring application UI to device, **62–67**, *63*, *64*, *66*, *67*
low-memory environment
checking return results of memory allocation, **57**
different types of memory allocation, **59–60**
EXE file size, **56–57**
handling WM_HIBERATE message, **60–61**
mass-allocating application memory, **58–60**

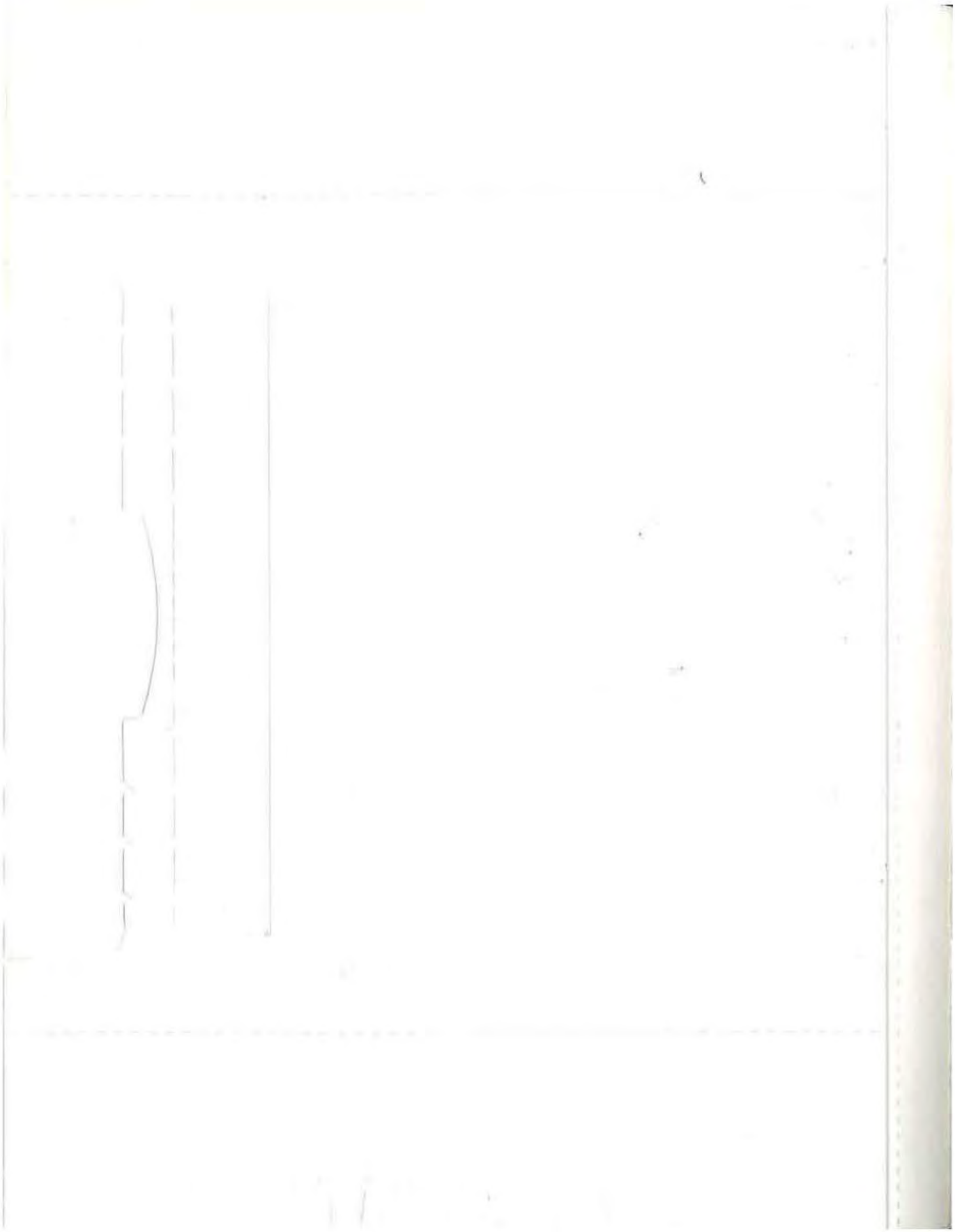
minimizing static variables, **56**
overview, **55–56**
reactivating application, **61**
overview, **42**
second commandment of, **55–61**
third commandment of, **61–72**
using Unicode Character set
declaring strings using Unicode types vs. char types, **43–46**, *44*, *45*
generic string types, **44–45**, *45*
overview, **42–43**
RTL (runtime library) functions, **47**, *47*
text literals, **46**
Unicode-specific string types, **44**, *44*
using with ANSI-based text files, **48–55**, *49*, *51*
wide strings, **43**
writing your own functions
fclose(), **88–89**
fgetc(), **89–90**
fgets(), **92–94**
FILE*-based functions, **81–83**
fopen(), **83–88**
fprintf(), **97–99**
fputs(), **94**
fputz(), **90–92**
fread(), **95–96**
fscanf(), **99–107**
fseek(), **108–109**
fwrite(), **96**
overview, **80–81**
wstrlen(), **95**
WSA-related functions, **327**
wsprintf(), **471**
wstrlen(), 84, **95**
_wtoi(), **449**
_wtol(), **449**

X

x86 Intel platform, **9**

Y

_y0(), **449**
_y1(), **449**
_yn(), **450**



About the CD

The companion CD-ROM contains all the code from the book, PDF versions of Appendices A and B, and valuable third-party utilities. For installation instructions, see the README.TXT file.

Code The CD contains all the code presented in the book, along with any resource files, VC++, VB, or Delphi project files, icons, etc. There are also additional code samples relating to topics only briefly touched on in the book. The source files are organized into directories named for each chapter, with subdirectories for each application. Please remember that if you copy any of these files to your hard drive manually, you must change the read-only file attribute before attempting to modify them. For your convenience, all of the code is also available in a single self-extracting archive.

Appendices The complete text of Appendices A and B (the C RTL and CE 2.0 API references) is provided in PDF format.

Shareware Shareware from DeveloperOne—a sampling of some of the best shareware utilities and applications for Windows CE.

Utilities Finally, you'll find several third-party samples, namely:

InstallShield for CE evaluation version	Evaluation version of InstallShield for Windows CE
Sybase Adaptive Server Anywhere sample application	Sample application demonstrating features of Adaptive Server Anywhere
Oracle Lite sample application	Sample application demonstrating features of Oracle's Windows CE products
The RDM/CE database engine	A third-party database engine for CE
Acrobat Reader	Viewer for PDF versions of Appendices A and B
Internet Explorer 4.01	A viewer for HTML files on the CD

Platforms The CD runs on Windows 95, Windows 98, Windows NT 4 Workstation.

WINDOWS CE
Developer's Handbook

Here's the book that helps every Windows programmer become a Windows CE pro!

The Windows CE Developer's Handbook is for experienced Windows developers who are ready to apply their skills to the rapidly expanding world of Windows CE. Inside, a CE expert offers an unflinching look at the realities of CE programming, including constraints on memory and processing power, a proliferation of device-specific capabilities, and a reduced API and runtime library. Can you meet these challenges? This book ensures that you can, providing in-depth coverage of the toughest tasks you'll face:

- Porting existing C/MFC/VB applications to Windows CE
- Working around CE devices' varying input and display capabilities
- Getting desktop applications to talk to CE devices
- Equipping your CE applications with support for key communications technologies: serial I/O, PCMCIA, infrared, and Winsock
- Building custom RAPI applications
- Creating CE installation and help files
- Using CE databases with C and MFC
- Taking complete advantage of each hardware platform's features
- Getting your application "logo-approved"
- Recreating file access functions missing from stdio.h
- Mastering the latest CE API calls
- Optimizing your code for the low-resource environment of CE

Featured on the CD: The enclosed CD contains all the sample code referred to in the book, a free ActiveX control for Windows CE, and evaluation software, including CE development utilities. You'll also find links to CE vendors, useful CE development Web sites, and PDF versions of the full documentation of CE's API and C runtime library functions.

ABOUT THE AUTHOR

Terence "Dr. CE" Goggin, a recognized Windows CE expert, is the author of numerous books and articles on Windows programming and the Internet. He is currently a consultant working on cutting-edge Windows CE solutions.

Coverage of E 2.0/2.01/2.11

Case Studies Crucial CE Strategies and Techniques

Master Undocumented Techniques for Solving the Toughest CE Development Challenges

Special Appendices Illustrate and Explain Every API Call and C Runtime Library Function—the Ultimate CE Desktop Reference

Bonus! Updates Online
www.sybex.com

For special bonus coverage, visit the author's Web site at www.DoctorCE.com

U.S. $49.99

ISBN 0-7821-2414-3
90000
EAN 9 780782 124149
UPC 0 25211 22414 4

SYBEX®

USER LEVEL	INTERMEDIATE/ADVANCED
BOOK TYPE	HOW-TO/REFERENCE
CATEGORY	PROGRAMMING

WINDOWS CE
Developer's Handbook
FT MEADE
MRC
QA 76
.76
.063
G636
1999
COPY 2