CD-Start per Mini-Programm

Selbstläufer

Nutzen Sie ein AutoRun-Programm für Ihre eigenen Projekte: Damit lassen Sie die per MP3-Playlist auf CD gebrannten Lieder automatisch in Ihrer favorisierten Folge abspielen. Oder Sie geben auf CD Bilder und Tabellen in Texten weiter, die Sie per Browser in einer HTML-Struktur übersichtlich aufbereitet präsentieren.

STEPHAN BEMME

Mit Windows 95 führte Microsoft die AutoRun-Funktion ein: Sobald Sie eine CD neu ins Laufwerk legen, startet eine Setup- oder Installationsroutine. Nahezu alle großen Software-Pakete nutzen diese elegante Technik. Sie können auch AutoRun in eigene Projekte einbauen. Denkbar ist, dass eine MP3-Playlist die auf CD gebrannten Lieder automatisch nach Ihrem Geschmack abspielt, oder dass eine HTML-Oberfläche den Inhalt Ihrer CD gefällig präsentiert.

Die gesamte Steuerung des Ablaufs übernimmt die Datei *AutoRun.inf*. Sie muss sich immer im Hauptverzeichnis der CD befinden. Da sie eine Textdatei ist, bearbeiten Sie die Einträge mit einem beliebigen Editor wie Notepad. Eine typische *AutoRun.inf* sieht etwa so aus:

```
[AutoRun]

icon=Autorun.ico
open=Autorun.exe index.html
```

500 EURO FÜR IHR C++-TOOL

PC Magazin sucht Ihr bestes C++-Programm für Windows: Schicken Sie uns per E-Mail Quellcode, Kompilat und Ihre Programmbeschreibung mit etwa 20000 Textzeichen. Ein veröffentlichtes Programm prämiert PC Magazin mit 500 Euro. Unsere Anschrift lautet: ethomas@pc-magazin.de.

Beachten Sie bitte die Hinweise im Impressum, Seite 165, linke Spalte unten. Ein Veröffentlichungsanspruch besteht nicht.

AUF CD

Das komplette Projekt finden Sie unter der Rubrik *Heft Add-ons/Programmierung/C++ Corner*.

AutoRun im Aufbau

Die Datei muss stets einen Abschnitt namens *[AutoRun]* enthalten. Alle Einstellungen, die in diesem Abschnitt festgelegt werden, sind optional. Es ist aber empfehlenswert, zumindest die Aktion, die beim Einlegen einer CD stattfinden soll, anzugeben. Dafür ist der Eintrag *open* zuständig. Alle direkt ausführbaren Programme, wie *exe*- und Batch-Dateien, können angegeben werden, relative Pfadangaben wie *\bin\setup.exe* sind ebenfalls zulässig.

Mit *icon* definieren Sie das Sinnbild, das der Explorer im Baumdiagramm an Stelle eines CD-Symbols zeigen soll. Es sind *.ico* oder *.bmp*-Dateien erlaubt. Bereits vorhandene Bilder wandeln Sie mit der Freeware Irfanview von Irfan Skiljan (www.irfanview.de) in diese Formate um. Eine eigene *.ico*-Datei legen Sie mit der Shareware IconEdit Pro (www.iconedit.com) an.

Klicken Sie mit der rechten Maustaste im Explorer auf das Symbol des CD-Laufwerks, erscheint das Kontextmenü. Es enthält bereits standardmäßig vordefinierte Einträge wie *Öffnen* und *Suchen*. Diesem Menü fügen die Befehle

```
shell\IrgendeinName=
    NameImKontextMenü
```

und

```
shell\IrgendeinName=
    command=Befehl
```

weitere CD-spezifische Einträge hinzu. Mit diesen Funktionen greifen Sie direkt auf Programme einer CD zu. So könnten Sie die Setup-Routine direkt über *open* zur Verfügung stellen oder ein Programm zur Analyse der Installation ein Systemanalyse-Programm im Kontextmenü anbieten.

AutoRun beschränkt sich nicht auf CDs. Nahezu alle Wechselmedien, wie Zip-Laufwerke, können damit umgehen. Mit Hilfe von Änderungen in der Registry lässt sich die Technik auch auf Festplatten, Netzlaufwerke und RAM-Disks anwenden, (wobei der Sinn aber fraglich ist). Interessant ist es, auf Disketten mit *Autorun.inf* zu experimentieren. Doch diese Tests laufen bei wiederbeschreibbaren CD-RWs wesentlich schneller ab, wenn Sie große Programme laden müssen. Weitere Details in der Microsoft Knowledge Base unter dem Code *Q136214*.

DAS BEISPIEL ZEIGT einen Klick in der Registry auf *HKEY_CLASSES_ROOT* und *.mp3*.

Denken Sie daran, dass ein Anwender *AutoRun* auch deaktivieren kann. Legen Sie daher eine Datei wie *Readme.txt* ins Hauptverzeichnis der CD, die die Vorgehensweise erklärt, falls die CD nicht automatisch startet.

Beliebige Dateien öffnen

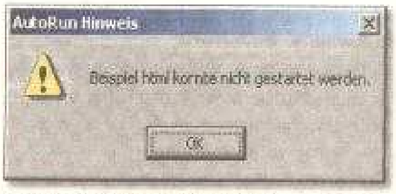
Auf Grund einer schlecht durchdachten Implementierung seitens Microsoft

können Sie mit dem *open*-Eintrag keine HTML-Seiten zeigen oder MP3s abspielen, weil nur *exe*- und Batch-Dateien erlaubt sind. Diese Lücke füllt das hier vorgestellte Programm, welches beliebige Dateien oder Programme anzeigen bzw. ausführen kann, die Sie als Parameter übergeben. Dies nutzen Sie in der *Autorun.inf* mit dem Eintrag:

```
open=Autorun.exe Homepage.html
```

So öffnen Sie die Seite *Homepage.html* im Standard-Browser. Das Programm besteht aus nur 60 Zeilen C++-Code, ohne Kommentare und die später im Text beschriebene Größenoptimierung wären es erheblich weniger. Um den Quellcode zu übersetzen, genügen ein beliebiger Windows-C++-Compiler und das Windows-SDK. Daher kommen Sie selb[illegible] zahlreichen Varianten des [illegible] Compilers aus, der unter folgenden Adressen verteilt wird:

www.cygwin.com
www.delorie.co[illegible]
www.mingw.com



SCHLÄGT *SHELLEXECUTE* FEHL, so öffnet sich dieses Fenster.

Visual Studio von Microsoft bietet zusätzlich eine Entwicklungsumgebung, die bei der Programmierung hilft. Als erstes legen Sie ein neues Projekt an: Im Menü *Datei/Neu* wählen Sie die *Win32-Anwendung* aus. Anschließend entscheiden Sie sich für *Ein leeres Projekt*. Diesem fügen Sie mit *Datei/Neu* im Reiter *Dateien* eine *C++ Quellcodedatei* hinzu. Es ist empfehlenswert, diese genau wie das Projekt zu benennen.

Nachdem Sie den Quelltext eingetippt haben, [illegible] *len/Aktive Konfiguration* festlegen von [illegible] lieren mit der Tastenkombination *[Strg-F5]* liegt im Verzeichnis *Release* die fertige *exe*-Datei.

Schuldzuweisung

Die Ursache dafür, dass in der *Autorun.inf* das Kommando *open* nur mit *exe*- und Batch-Dateien umgehen kann, liegt bei den Windows-Programmierern. Sie haben *open* intern mit Hilfe des Win32-API-Aufrufs *CreateProcess* umgesetzt. Damit kann *Autorun.inf* nicht umgehen – nur Programmiersprachen wie C.

Der Aufruf funktioniert bei ausführbaren Dateien mit beliebigen Parametern. Leider kann er mit anderen Dateitypen nichts anfangen. Verwenden Sie statt *CreateProcess* den Befehl *ShellExecute*, so können Sie neben *exe*- und [illegible] Dokumente übergeben. Das sehen Sie in der Registry an der Dateiendung, welche mit einer Anwendung assoziiert ist. Wenn ja, startet diese. So lassen sich HTML-Dokumente im [illegible] Internet Explorer aufrufen und anzeigen. Natürlich

startet auch Opera oder der Netscape Navigator – sofern installiert.

In der Registry verfolgen Sie den Weg, den *ShellExecute* geht, wenn Sie *regedit.exe* starten. Vorsicht: Änderungen der Registry können Windows beschädigen! Im Bereich *HKEY_CLASSES_ROOT* sind alle dem System bekannten Dateiendungen aufgezählt. Klicken Sie z.B. *.mp3* an, so sehen Sie Einträge ähnlich denen im Bild auf der ersten Seite des Beitrags.

Wichtig ist der Eintrag *Standard*, welcher auf *Winamp* verweist. Denn der Wert *Winamp.File* ist ebenfalls ein Schlüssel in der Registry. Unter *HKEY_CLASSES_ROOT\Winamp.File* findet sich der Unterschlüssel *shell\open\command*. Dieser gibt an, welches Programm starten soll, wenn Sie eine *mp3*-Datei öffnen. Falls eine Dateiendung nicht auf einen Eintrag verweist, der einen Unterschlüssel *shell\open\command* besitzt, so lässt sich die Datei nicht mit *ShellExecute* öffnen.

Der API-Befehl *ShellExecute* ist in allen Windows-Versionen vorhanden. Um ihn zu [illegible] grammen, die Header-Datei *windows.h* einzubinden. Der Befehl arbeitet, wenn Sie im Start-Menü unter *Ausführen ...* einen Programm- oder Dateinamen eingeben oder im Explorer auf einen Dateinamen doppelt klicken. Um *ShellExecute* richtig einzusetzen, müssen Sie sechs Parameter vorbereiten.

Da Sie auf keine Nachrichten warten, die das mit *ShellExecute* gestartete Programm empfängt, setzen Sie *hwnd* auf *NULL*. Beachten Sie den Parameter *lpOperation*. Unser Programm arbeitet theoretisch mit fünf Operationen:

- *open* (– öffnen, anzeigen bzw. starten).

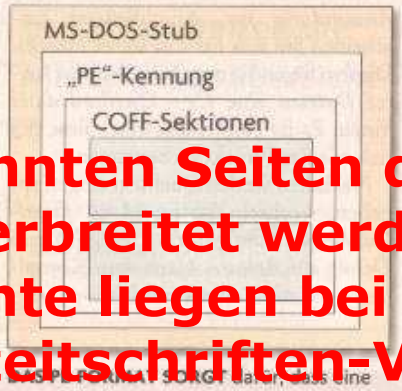
Daneben funktionieren weitere Parameter:

- *edit* (– bearbeiten),
- *explore* (– Explorer öffnen, wobei *lpFile* ein Verzeichnis sein muss),
- *find* (– Suchfenster aufklappen) oder
- *print* (– drucken).

Nicht jede dieser Operationen ist stets verfügbar, so können Sie keine *exe*-Dateien drucken. Der Parameter *lpOperation* wird auch auf *NULL* gesetzt, womit die jeweilige Standard-Operation ausgeführt werden soll, also fast immer *open*.

Dateinamen kunstgerecht extrahieren

Den Dateinamen *lpFile* und seine Parameter *lpParameter* extrahieren Sie aus der Kommandozeile unseres Programms. Dabei zählt alles, was vor dem ersten Leerzeichen steht, als Datei, der



Rest sind die Parameter. Schwieriger ist es, mitten in Verzeichnissen oder Dateinamen auftretende Leerzeichen korrekt abzuarbeiten. So muss eine im Verzeichnis *Eigene Dateien* vorhandene Datei *Dokument.doc* mit Anführungsstrichen geschrieben werden, da sie sonst fälschlich als eine Datei *Eigene* mit dem Parameter *Dateien\Dokument.doc* interpretiert würde. Fast die Hälfte des Quellcodes trennt daher Kommando und Parameter. Die Idee ist, dasjenige Leerzeichen zwischen Kommando und Parameter durch eine *Null* zu ersetzen, das in C für das Ende einer Zeichenkette steht.

Das aktuelle Verzeichnis soll auch das Verzeichnis sein, in dem die Datei ausgeführt wird. Daher ist der fünfte Parameter von *ShellExecute* ebenfalls *NULL*, weshalb das Hauptverzeichnis der CD gewählt wird.

Als Anzeigemodus dient die Konstante *SW_SHOWNORMAL*, um das jeweilige Anzeigeprogramm als Standard einzustellen.

Der Befehl *ShellExecute* gibt einen Wert vom Typ *HINSTANCE* zurück. In Wahrheit verbirgt sich dahinter ein 32-Bit-Integer. Das gibt das SDK von Microsoft zu und rät daher zu einer Typ-Umwandlung (Cast). Sollte dabei eine Zahl größer als 32 entstehen, so war die Operation erfolgreich. In allen anderen Fällen identifizieren Sie Fehler an Hand des Rückgabewertes. Das Programm verzichtet darauf, da in 99 Prozent aller Fälle die Ursache darin liegt, dass die angegebene Datei nicht existiert oder gesperrt ist.

Falls *ShellExecute* fehlschlagen sollte, öffnet sich ein Fenster mit dem Win32-API-Befehl *MessageBox*, das den Anwender darüber informiert und einen *OK*-Knopf zeigt (Parameter *MB_OK*). Dazu [illegible] das Ausrufezeichen (MB_ICONWARNING) aus. Sie können sich auch für ein Stopp-Schild (MB_ICONSTOP) oder das Informationssymbol (MB_ICONINFORMATION) entscheiden.

Inzwischen hat Microsoft seinen Fehler berichtigt. Ab Windows 2000 setzen Sie an Stelle von *open* in der Datei *Autorun.inf* auch das Kommando *shellexecute* ein.

Allerdings startet diese CD dann nicht automatisch auf älteren Windows-Rechnern, was den Gebrauch von *shellexecute* einschränkt. Anwender müssten dann noch mit dem Internet Explorer 5 einige Dateien des Betriebssystems austauschen damit Windows 95, 98 und ME doch wieder mit dem Kommando *shellexecute* arbeiten können. Besser verwenden Sie also das *Autorun*-Progamm.

Das *PE*-Format

Die kompilierte *exe*-Datei ist als Release-Version 40 KByte, als Debug-Version 168 KByte groß. Selbst wenn Sie im Menü *Projekt/Einstellungen* im Reiter *C/C++ Größe minimieren* wählen, kann der Compiler das Programm nicht verkleinern. Wieso ist die Datei so groß, wo der Quellcode kaum länger als eine Seite ist ?

Die Übergröße verursacht nicht der Compiler. Die von ihm erzeugte *OBJ*-Datei enthält den in Maschinencode umgesetzten Quelltext und ist dabei in etwa

DIE SECHS PARAMETER VON *SHELLEXECUTE*

Parameter	Datentyp	Bedeutung
hwnd	HWND	Handle des aufrufenden Programms
lpOperation	LPCTSTR	Zeichenkette, auszuführende Operation
lpFile	LPCTSTR	Dateiname
lpParameters	LPCTSTR	Parameter, falls *lpFile* eine ausführbare Datei ist
lpDirectory	LPCTSTR	Verzeichnis
nShowCmd	INT	Anzeige-Modus

ein KByte groß (Release-Version). Untersuchen Sie dies per Hex-Editor, so entdecken Sie Zusatzinformationen für den Linker. Der Compiler würde den Quelltext in nur wenige Hundert Bytes Maschinencode verwandeln. Doch der Linker ist dafür verantwortlich, dass die *exe*-Dateien so groß werden.

Unter Win32 (Windows 95, 98, ME, NT, 2000 und XP) sind *exe*-Dateien nach dem *PE*-Format (Portable Executable) aufgebaut, stark abweichend von *exe*-Dateien unter DOS. Falls jemand eine *PE*-exe unter DOS startet, sorgt ein sogenannter Stub dafür, dass Sie die Bildschirmausgabe

```
This program cannot be run in DOS
mode.
```

sehen, das System aber nicht abstürzt. Im Hex-Editor sehen Sie diese Zeichenkette innerhalb der ersten paar Bytes des Programms.

Erst danach beginnt die *PE*-Datei, gekennzeichnet durch *PE* [illegible] gefolgt vom *COFF*-Format (Common Object File Format), das aus mehreren Sektionen besteht. Die genauen Details lesen Sie unter http://msdn.microsoft.com/. Der Linker sorgt dafür, dass alle Sektionen in der exe-Datei an Speicheradressen beginnen, die ohne Rest durch 4096 teilbar sind. Dadurch kann Windows die Programme schneller starten, da sie später auch im Hauptspeicher auf 4096-Byte-Grenzen ausgerichtet sind. Erst mit Windows 98 führte Microsoft dieses so genannte 4096-Byte-Alignment ein, vorher reichten 512 Bytes. Teilen Sie dem Visual C++ Compiler mit, dass Sie das *alte* 512-Byte-Alignment wünschen, schrumpft unser Programm auf 31 KByte.

Wenn Sie das kompilierte Programm später starten, dauert das Laden in den Hauptspeicher ein paar Tausendstelsekunden länger. Dieser Zeitgewinn lässt sich jedoch nur mit aufwändigen Messmethoden nachweisen. Sie erreichen die Änderung mit einer zusätzlichen Zeile:

```
// 4KByte-Ausrichtung
// deaktivieren
#pragma comment
 (linker, „/OPT:NOWIN98")
```

Standardbibliothek auslagern

Vergleichen Sie mehrere *exe*-Dateien, so erkennen Sie viele Ähnlichkeiten. Sie binden alle eine Standardbibliothek ein, die unter anderem für Funktionen wie [illegible] auf und zum Beispiel den mathematischen Coprozessor initialisiert. Ein großer Teil dieses Codes ist für das AutoRun-Programm nicht notwendig. Dennoch wird er vom Linker hinzugefügt.

Microsoft hat erkannt, dass die Standardbibliothek in allen C-Programmen, die Sie mit Visual C++ kompilieren, identisch vorhanden ist. Auf jedem Windows-System findet sich daher eine Datei namens *msvcrt.dll* (MicroSoft Visual C RunTime) im Verzeichnis *Windows\System* (Windows 95, 98, ME) bzw. *Windows\System32* (Windows NT, 2000, XP). Diese DLL binden Sie ebenfalls mit nur einer Zeile Code ein, allerdings müssen Sie zuvor die Standardbibliothek deaktivieren.

Da sie für Debug- und Release-Version unterschiedlich ist, brauchen Sie diese drei Zeilen:

```
// Standardroutinen nicht
//statisch einbinden ...

#pragma comment (linker,
        „/NODEFAULTLIB:libc.lib")
#pragma comment(linker,
        „/NODEFAULTLIB:libcd.lib")
// ... sondern MSVCRT.DLL
// MS Visual C RunTime nutzen
#pragma comment
              (lib, „msvcrt.lib")
```

Das Programm schrumpft auf 3584 Bytes! Wollen Sie es weiter minimieren, so könnten Sie so genannte Executable-Komprimierer wie die UPX-Freeware http://upx.source forge.net verwenden, die eine *exe* packen, aber trotzdem ausführbar lassen. Diese Komprimierer scheitern bei geringen Dateigrößen, da sie erst ab etwa 10 KByte sinnvoll einsetzbar sind.

Matt Pietrik veröffentlichte in der Ausgabe 01/2001 des MSDN Magazine die *LibCTiny*. Sie ersetzt die Standardbibliothek und verzichtet auf *MSVCRT.DLL* (Details unter:) http://msdn.microsoft.com. Sie kann jedoch auch die 3584 Bytes nicht unterbieten. Vermutlich lässt sich eine kleinere *exe*-Datei nur noch mit Assembler erreichen. Viel Spaß dabei! ET

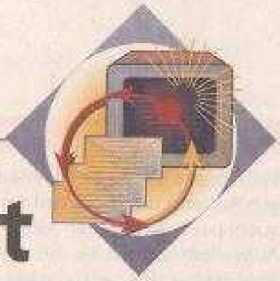
autorun.cpp

```
// autorun.cpp, Windows-Quellcode
#include <windows.h>
// für "sprintf"
#include <stdio.h>


// 4k-Ausrichtung deaktivieren
#pragma comment(linker, "/OPT:NOWIN98")
// Standardroutinen nicht statisch einbinden ...
#pragma comment(linker, "/NODEFAULTLIB:libc.lib")
#pragma comment(linker, "/NODEFAULTLIB:libcd.lib")
// ... sondern aus MSVCRT.DLL beziehen
#pragma comment(lib, "msvcrt.lib")

// für "GetCommandLine"
#pragma comment(lib, "kernel32.lib")
// für "MessageBox"
#pragma comment(lib, "user32.lib")
// für "ShellExecute"
#pragma comment(lib, "shell32.lib")


int WINAPI WinMain(HINSTANCE,HINSTANCE, LPSTR pCmdLine, int)
{
 // Parameter extrahieren
 LPSTR pParameter = pCmdLine;

 bool bQuoted = false;
 while ((*pParameter != ' ' || bQuoted) && *pParameter != 0)
 {
  // in Hochkommata eingeschlossen ?
  if (*pParameter == '<\\>"')
   bQuoted = !bQuoted;

  // weiter zum nächsten Zeichen
  pParameter++;
 }

// Anwendung und Parameter trennen
if (*pParameter != 0)
{
 *pParameter = 0;
 pParameter++;
}
else
 // keine Parameter gefunden
 pParameter = NULL;


// Anwendung/Dokument öffnen
   if ((int) ShellExecute(NULL, NULL, pCmdLine, pParameter,
                          NULL, SW_SHOWNORMAL) <= 32)
   {
       // Datei konnte nicht ausgeführt/angezeigt werden
       char sz[256];
       sprintf(sz, "%s konnte nicht gestartet werden.",
pCmdLine);
MessageBox
 (NULL, sz, "AutoRun Hinweis", MB_OK | MB_ICONWARNING);

       // Fehler
       return 1;
   }

   return 0;
}
```

autorun.cpp beschränkt sich auf wenige Zeilen Quellcode.

Wir lösen Ihre Hard- und Software-Probleme

Mit Rat und Tat



C++-CORNER, 8/02, S. 156

Doppelfehler

In der Beschreibung zum C++-Programm *Autorun* steht, man solle Leerzeichen bei *Verzeichnis aufrufen* extrahieren. Leider funktioniert die beschriebene Methode nicht: Sobald ein Leerzeichen im Aufruf von Verzeichnissen auftaucht, wird folgende Anweisung ungültig und verl[illegible]:

```
while( (*pParam != ' ' || bGetIt)
&& *pParam != 0)
```

Vielleicht hätten Sie in der C++-Corner erwähnen sollen, dass man schon beim Programmstart einen Parameter übergeben muss, um die Anweisung

```
while( [illegible]
&& *pParam != 0)
```

zu testen. Dies gelingt über *[Alt-F7]/Debug/Program arguments/C:\Eigene Dateien\Doc.txt.*

SEBASTIAN FIEBIGER (VIA E-MAIL)

ANTWORT DER REDAKTION: *Die Heft-CD 08/02 enthält im Verzeichnis* Heft Add-ons/Programmierung/C++ Corner *die Datei* c_corner.exe *mit dem komprimierten Quellcode. Entpacken Sie diese, bekommen Sie zweimal die Datei* Autorun.cpp: *einmal direkt im Verzeichnis, in welchem Sie entpackt haben, und ein weiteres Mal im Unterverzeichnis* programm. *Erstere* Autorun.cpp *haben wir erzeugt, indem wir die Datei von ASCII nach ANSI konvertierten. Der Fehler in dieser Datei hat sich ins abgedruckte Listing, Zeile 32, S. 159, eingeschlichen: Der Rückstrich \ wurde zur Zeichenkette < \\ >. Arbeiten Sie also mit der Datei* Autorun.cpp *aus dem Unterverzeichnis* programm. *Diese Zeile berücksichtigt, ob das Argument in Anführungszeichen steht.*

Die Windows-Konvention verlangen dies, wenn Parameter Leerzeichen enthalten sollen. Dies testen Sie selbst, indem Sie vom Windows-Desktop über Start/Ausführen *eine Datei aus dem Verzeichnis* Eigene Dateien *wählen. Dort [illegible] Windows [illegible] che automatisch hinzu, da es sonst die [illegible] det, [illegible]*

```
„C:\Eigene Dateien\Test.html"
...
C:\Eigene Dateien\Test.html
```

Das Wurzelverzeichnis von Windows [illegible] Anführungszeichen nicht. Dort genügt

```
C:\Test.html
```

Ihr Hinweis mit dem Parameter beim Programmstart ist richtig.

STEPHAN BRUMME/ET

SIE FRAGEN, WIR ANTWORTEN

Schildern Sie uns Ihre Hard- und Software-Probleme: Jede E-Mail und Zuschrift ist uns willkommen. Nennen Sie bitte Ihre Adresse inklusive Telefonnummer, und geben Sie Ihre Rechnerkonfiguration an.
Unsere Adresse: WEKA Computerzeitschriften-Verlag, PC Magazin, Stichwort: *Support*, Gruber Str. 46a, 85586 Poing; E-Mail: praxis@pc-magazin.de

RECOVERY-CD

Bequeme Sicherung

Ich möchte eine Recovery-CD herstellen, welche meine Festplatte beim Einlegen der CD formatiert und das auf der CD befindliche Image (Windows 98/2000 oder XP) wieder installiert, inklusive aller Registry-Einträge und wichtiger Programme.

Gibt es solche Werkzeuge zu kaufen? Oder kann ich diese Recovery-CD selber machen?

ROLAND BLÄTTLER (VIA E-MAIL)

ANTWORT DER REDAKTION: *Im Prinzip brennen Sie eine solche CD selbst. Alle gängigen CD-Brennprogramme haben eine Funktion, mit der Sie den Inhalt so auf eine CD brennen, dass die CD-ROM wie eine Startdiskette funktioniert. Aber um ein Image herzustellen, benötigen Sie sowieso ein Image-Programm. Marktführer sind Norton Ghost von Symantec (www.symantec.de) und DriveImage von Powerquest (www.powerquest.de). Beide Programme können bootfähige Rettungs-CDs mit einem Festplatten-Image herstellen.*

WOLFGANG NEFZGER/ET

VB: ANSI IN ASCII UMWANDELN

DOS-Format speichern

Wie kann ich in Visual Basic einen Text im DOS-Format (ASCII) speichern? VB speichert im ANSI-Format, das ein DOS-Programm nicht lesen kann.

HUNG NGUYEN-CONG (VIA E-MAIL)

ANTWORT DER REDAKTION: *Am einfachsten wandeln Sie ANSI- in ASCII-Zeichenketten durch die Windows API-Funktion* CharToOem *um. Diese deklarieren Sie folgendermaßen:*

```
Declare Function CharToOem Lib
„user32" Alias „CharToOemA"
(ByVal Src As String, ByVal Dest
As String) As Long
```

Anschließend können Sie die ASCII- zu einer ANSI-Zeichenkette in einer sequenziellen Datei sichern:

```
...
'freie Dateinummer ermitteln
Dim Dnr As Integer
Dnr = FreeFile
'[illegible]
Open [illegible] For Output As Dnr
'Zeichenkette umwandeln/speichern
Dim ansi As String
Dim ascii As String
ansi = „äöpdß"
'Zielzeichenkette initialisieren
ascii = Space(Len(ansi))
'Konvertierung ANSI zu ASCII
dummy&=CharToOem(ansi, ascii)
Print #Dnr, ascii
Close Dnr
...
```

Wichtig ist, dass die ASCII-Zielzeichenkette noch vor der Zeichenkettenformatierung auf die erforderliche Ausgangszeichenkettenlänge gebracht wird. Mit der API-Funktion OemToChar *können Sie entsprechend ASCII- in ANSI-Zeichenketten umwandeln.*

DIPL.-ING. ANDREAS MASLO/ET

JAVASCRIPT: IP-ADRESSE

Spur im Nebel

Kann ich die IP-Adresse eines Besuchers per JavaScript ermitteln, der auf meiner Homepage war?

ESTER PADENSKI (VIA E-MAIL)

ANTWORT DER REDAKTION: *Mit JavaScript allein kann das nicht gelingen. Aber in Verbindung mit SSI (Server Side Includes) wäre es möglich, die IP des Besuchers auszulesen und innerhalb eines Scripts zu verarbeiten. Die Adresse ließe sich vielleicht sogar an ein verstecktes Formular-Element weiterreichen. Damit könnte man an den Besucher dieser Site sogar eine E-Mail versenden.*

MATHIAS SIMMACK/ET