Haben Sie Lust auf ein Spiel?

Haben Sie jemals Kindern beim Spielen zugesehen?

Es ist erstaunlich, wie sich Kinder in ein Spiel vertiefen können. Spaß ist eine ernste Sache, wenn man ein Kind ist.

Bemerkenswert ist die Aussage der meisten Kinderpsychologen, dass Kinder gerade beim Spielen eine Menge lernen.

Es scheint, dass uns diese Form des Lernens verloren gegangen ist. Das Lernen wird zu einer Aufgabe – zunächst, weil wir gute Noten bekommen möchten, später, weil wir auf dem Laufenden bleiben müssen, wenn wir in der heutigen Zeit mithalten wollen. Und so geht der Spaß am Lernen nach und nach verloren. Das Spielen wird ersetzt durch endlose, langweilige Vorträge und trockene, theoretische Lehrbücher. Wer sagt, dass Informationen in dieser Form vermittelt werden müssen? Wer sagt, dass technische Handbücher ernst und trocken sein müssen? Ich bin dagegen!

Und ich bin auch dagegen, dass Lernen harte Arbeit sein muss!

In gewisser Weise haben wir Programmierer Glück. Die meisten von uns erhalten von Zeit zu Zeit die Gelegenheit, neue Technologien auszuprobieren, auch wenn sich die so investierte Zeit bei einem vorgegebenen Projekt nicht immer rechtfertigen lässt bzw. dem Kunden nicht in Rechnung gestellt werden kann. Wir verbringen Stunden damit, herauszufinden, wie etwas funktioniert oder finden neue, tolle Methoden heraus, wie man etwas erledigen kann, auch wenn wir dieses Wissen später nie anwenden. Man kann dies »Lernen«, »Forschen« oder »Testen« nennen, aber, unter uns gesagt, handelt es sich doch eigentlich um eine Spielerei. Es macht Spaß. Und genau auf diese Weise lernt es sich am besten.

Also, lassen Sie uns gemeinsam lernen.

Lassen Sie uns spielen.

Aufbau dieses Buches

Der Aufbau dieses Buches unterscheidet sich von den Büchern, die ich bisher geschrieben habe. Ich kann mich ehrlich gesagt nicht daran erinnern, vorher bereits etwas Ähnliches gesehen zu haben. Das Buch besteht aus drei Teilen mit unterschiedlichen Zielen. Darüber hinaus wurde dieses Buch **nicht** so geschrieben, dass es von Anfang bis Ende gelesen werden müsste.

Teil I: Die Puzzle[1]

Jedes »Kapitel« in Teil I umfasst ein kleines Puzzle. Die ersten Puzzle (im ersten Buchabschnitt) sollten zu einer nur mittelschweren Frustration führen, selbst für solche VB-Programmierer, die sich im Anfangsstadium befinden und nur über begrenzte Erfahrung bei der API-Programmierung verfügen. Der Schwierigkeitsgrad der Puzzle erhöht sich, sodass die Puzzle am Ende des Buches – Abschnitte 4 und 5 – selbst für Experten eine Herausforderung darstellen sollten. Obwohl die meisten der Puzzle ein Codebeispiel umfassen, das nicht funktioniert und berichtigt werden muss, sollten Sie sich dennoch auf einige Überraschungen gefasst machen.

Wenn Sie Probleme haben, finden Sie Hilfe in Anhang A, »Hinweise« (siehe auch die nachstehenden Hinweisinformationen).

Teil II: Die Lösungen

Hier finden Sie die Lösungen für die Puzzle in Teil I. Jede Lösung umfasst nicht nur den richtigen Code, sondern auch eine detaillierte Erläuterung des vorliegenden Problems sowie des entsprechenden Lösungsansatzes. Obwohl ich verstehen kann, dass die Versuchung, direkt zur jeweiligen Lösung zu wechseln, sehr groß sein kann, sollten Sie dennoch versuchen, einige Zeit über jedes Puzzle nachzudenken, bevor Sie sich die Lösung ansehen. Auf diese Weise ist der Lerneffekt höher, und Sie werden das Erlernte besser behalten.

Teil III: Die Tutorien

Diese detaillierten Tutorien umfassen Themen, die sich auf den API-Funktionsaufruf aus Visual Basic beziehen. Einige dieser Themen wurden speziell ausgesucht, um einige Wissenslücken zu schließen, die bei der Ausbildung zum Visual Basic-Programmierer erfahrungsgemäß entstehen. Andere Themengebiete beinhalten Beispiele für fortgeschrittene Programmierer, die sich für den Puzzle-Ansatz nicht eignen. Wieder andere der Tutorien enthalten Informationen oder Verfahrensweisen, von denen ich nicht wusste, wo ich sie denn einordnen sollte.

Die Anhänge

Anhang A: Hinweise

Die Lösungen enthalten sämtliche Informationen, die zur Lösung der Puzzle in diesem Buch benötigt werden. Da Sie dieses Buch lesen, ist Ihnen sehr wahrscheinlich bewusst, dass es viele Fälle gibt, in denen ein Programmierproblem nicht einfach

1. Anmerkung des Verlags: Engl. »Puzzles« wurde mit »Puzzle« übersetzt.

mit Hilfe eines Buches gelöst werden kann. Sie werden auch andere Informationsquellen heranziehen müssen, um die Antwort auf eine Frage zu erhalten. Dieses Buch wurde beispielsweise **nicht** konzipiert, um als Referenz bei der Beantwortung von Fragen im Hinblick auf spezielle API-Funktionen zu dienen.

Wenn Sie die Lösung zu einem Problem nicht wissen, verwenden Sie Anhang A, »Hinweise«. Dort finden Sie Hinweise auf weitere Informationsquellen.

Im Folgenden werden einige Standardinformationsquellen genannt, die Sie stets zu Rate ziehen sollten:

- **The Win32 SDK or MSDN (Microsoft Developer's Network Library oder http://www.microsoft.com):**
  Hier finden Sie die aktuelle C/C++-Dokumentation für die fragliche API bzw. die aktuellste Bugliste. Visual Basic enthält eine eingeschränkte Version der MSDN-Bibliothek, die wiederum die Win32 SDK-Dokumentation umfasst; Sie verfügen also bereits über die benötigten Informationen.
- **Die C-Headerdateien:**
  Hierbei handelt es sich um jene C-Headerdateien, die von C++ Windows-Entwicklern verwendet werden und gleichzeitig als wichtigste Referenz in Bezug auf die richtigen Funktionsbeschreibungen dienen. Der vollständige Satz der Win32 SDK-Headerdateien ist, dank freundlicher Unterstützung von Microsoft, auf der Begleit-CD-ROM zu diesem Buch enthalten.
- **Die Visual Basic-Dokumentation:**
  In dieser Dokumentation sind einige Informationen zum API-Funktionsaufruf enthalten.
- **Kapitel 1 bis 4 meines Buches »Dan Appleman's Visual Basic Programmer's Guide to the Win32 API«** (SAMS, ISBN 0-672-31590-4)**:**
  Diese Kapitel enthalten grundlegende Informationen zu Windows und dem API-Funktionsaufruf, deren Kenntnis zur Lösung der Puzzle in diesem Buch von höchster Wichtigkeit sind.

Zusätzlich enthalten die Hinweise in Anhang A Verweise auf eine oder mehrere der folgenden Informationsquellen, die bei der Lösung der Puzzle ebenfalls nützlich sein können:

Mein Api-Buch (siehe oben), Anhang D:
Ich habe in vielen Fällen auf dieses spezielle Kapitel in meinem API-Buch verwiesen, da es sich auf die im Puzzle verwendete API bezieht. Obwohl Ihnen das Kapitel auch nur zur Puzzlelösung dienen mag, stellt es doch darüber hinaus nützliche Hintergrundinformationen zu den fraglichen Funktionen bereit und dient als weitergehende Informationsquelle.

- **Verweise auf die »Zehn API-Gebote«**
  Hierbei handelt es sich um einen Verweis auf die »Zehn Gebote für die sichere API-Programmierung« (nachfolgend aufgeführt). Diese können als Richtlinien bei der Lösungsfindung für nahezu jedes API-Problem herangezogen werden.
- **Tutorien**
  Hierbei handelt es sich um einen Verweis auf ein entsprechendes Tutorium (in Teil III dieses Buches), in dem die der Puzzlelösung zugrundeliegenden Konzepte erläutert werden. Diese Hinweise sind besonders hilfreich für Programmierer mit geringerer Erfahrung und eignen sich gerade zur Lösung der leichteren Puzzle.
- **Mein anderes Buch »Dan Appleman's Developing COM/ActiveX Components with Visual Basic 6.0: A Guide to the Perplexed«** (SAMS, ISBN 1-56276-576-0)**:** Was hat ein Buch über ActiveX mit der Win32-API zu tun? Eine ganze Menge, wenn Sie bedenken, dass mit der OLE- oder ActiveX-Technologie von Microsoft mit Hilfe von DLLs implementiert wird, die praktisch über Hunderte von Funktionen verfügen.

Anhang B: Häufig gestellte Fragen

In diesem Buch werden zahlreiche Methoden für den API-Funktionsaufruf aus Visual Basic vorgestellt. Aufgrund des Buchaufbaus sind diese Techniken über die verschiedenen Puzzle und Tutorien verteilt. Daher habe ich mich entschieden, den Lesern eine Liste mit Querverweisen zur Verfügung zu stellen, aus der zu entnehmen ist, an welcher Stelle des Buches eine bestimmte Methode behandelt wird.

Wie die meisten technischen Bücher verfügt auch dieses über einen Index, damit Informationen zu bestimmten Themen schnell aufgefunden werden können. Da es sich bei jedem Index jedoch um nichts weiter als ein Suchsystem nach dem Schlüsselwortprinzip handelt, weiß jeder, der bereits mit einer Internetsuchmaschine gearbeitet hat, dass diese Suchsysteme nur begrenzte Funktionen besitzen.

Dieser Anhang enthält neben dem Index eine Liste mit häufig gestellten Fragen. Hierbei handelt es sich de facto um ein Verzeichnis der API-Techniken, die in Kategorien unterteilt wurden.

Anhang C: Die APIGID32.DLL-Bibliothek

Dieser Anhang enthält eine Dokumentation für die DLL **apigid32.dll**, eine Dynamic Link Library, die über nützliche Funktionen zur Arbeit mit API-Funktionen von Visual Basic aus verfügt. Für diejenigen, die daran interessiert sind, C++-DLLs für die Verwendung mit Visual Basic zu schreiben, wurde der Quellcode dieser DLLs beigefügt.

Verwendung der CD-ROM

Die wichtigste Datei auf der Begleit-CD-ROM zu diesem Buch stellt die Datei **pzl1.hlp** im Stammverzeichnis dar. Doppelklicken Sie einfach auf die Datei, um den Hilfebildschirm zu starten. Die Hilfe umfasst Folgendes:

- Die aktuellsten Installationsanmerkungen und Berichtigungen, die in letzter Minute vorgenommen wurden, also das Äquivalent zur traditionellen Readme-Datei
- Links zum Installationsprogramm, mit denen benötigte Steuerelemente und DLLs zur Ausführung des Beispielprogramms installiert werden
- Eine halbstündige Videopräsentation
- Eine Liste der CD-ROM-Hauptverzeichnisse sowie deren Inhalt
- Den Onlinekatalog von Desaware

Der Beispielcode befindet sich in den Verzeichnissen **SourceVB6** und **SourceVB5**. Das Verzeichnis **SourceVB6** enthält den VB6-Quellcode, der als Referenz für den Quellcode in diesem Buch dient. Das Verzeichnis **VB5** enthält die VB5-Versionen des Beispielprogramms. Das (in den Puzzlen und Lösungen verwendete) Codeverhalten basiert jedoch auf VB6. Die VB5-Versionen wurden als zusätzliche Hilfe beigefügt und sollten das gleiche Verhalten aufweisen, eine Garantie hierfür kann ich jedoch nicht übernehmen.

Im Verzeichnis **VB6** befinden sich drei Unterverzeichnisse, **Puzzle**, **Lösungen** und **Tutorien**. Jedes dieser Verzeichnisse enthält weitere Unterverzeichnisse für die einzelnen Puzzle oder Tutorien. Das Verzeichnis **Puzzle** enthält den Beispielcode, der das im Puzzle beschriebene Problem zeigt. Im Verzeichnis **Lösung** befinden sich die Lösungen für jedes Puzzle. Im Verzeichnis **Tutorien** ist der Beispielcode für die Tutorien gespeichert.

Das Verzeichnis **Cheaders** enthält, dank der freundlichen Unterstützung von Microsoft, die C-Headerdateien von Microsoft, die unter dem Programm **Open Tools** zur Verfügung gestellt werden. In diesen Headerdateien finden Sie die Werte von Konstanten sowie Datentypinformationen.

Weitere Informationen zum Inhalt der CD-ROM finden Sie in der Datei **pzl1.hlp**.

Info zu Betriebssystemen

Ich weiß nicht, welches Betriebssystem Sie verwenden. Selbst wenn Sie es mir sagen würden, würde ich immer noch nicht wissen, welches Betriebssystem Sie einsetzen.

Als dieses Buch in Druck ging, haben Sie wahrscheinlich eines der folgenden Betriebssysteme verwendet: Windows 95, Windows 98, Windows NT 3.51, Windows NT 4.0 oder die Betaversion von Windows 2000.

Dies bedeutet, dass ich jedes der Puzzle unter einem dieser fünf Betriebssysteme hätte testen müssen, um sicher sagen zu können, dass die beschriebenen Lösungen tatsächlich mit den Ergebnissen übereinstimmen, die Sie erhalten. Aber selbst diese Tests würden aus verschiedenen Gründen nicht ausreichen:

- Sie verfügen vielleicht über die Originalversion von Windows 95, oder Sie haben ein Service Pack installiert, oder es handelt sich um die OSR1- oder OSR2-Version.
- Ihre NT 3.51-Installation kann über eines von fünf verschiedenen Service Packs verfügen.
- Ihre NT 4.0-Installation kann über eines von sechs verschiedenen Service Packs verfügen.

Selbst wenn ich wüsste, welche Windows 95-Version Sie verwenden oder welche NT-Service Packs Sie installiert haben, auch dies würde noch nicht ausreichen, da bei jeder Installation eines Microsoft-Pakets – sei es nun Visual Studio (in sämtlichen Versionen und einschließlich der jeweiligen Service Packs), Internet Explorer (mit monatlichen Änderungen) oder Microsoft Office – unter Umständen neue Systemkomponenten installiert werden.

Was sollte ein Autor also tun?

Wenn wir bei Desaware Software veröffentlichen, testen wir alle gängigen Betriebssystemversionen mit jeweils einer Grundinstallation und dem aktuellsten Service Pack. Anschließend starten wir unsere Anwendungen oder Komponenten mit Hilfe unseres eigenen Produkts (VersionStamper) mit Selbstdiagnosefähigkeiten aus, um so nachvollziehen zu können, welche Fehler auf einem Kundensystem vorliegen (damit in bestimmten Fällen automatisch Komponenten von unserer FTP-Site heruntergeladen werden können).

Ein Buch kann sich jedoch nicht selbst aktualisieren, und eine gedruckte Seite ist nicht in der Lage, Ihr System zu scannen.

Deshalb wird in diesem Buch angenommen, dass Sie Windows NT 4.0 verwenden. Windows NT stellt keine schlechte Wahl dar, wenn Sie genauer darüber nachdenken – die meisten professionellen Entwickler arbeiten statt mit Windows 95/98 mit Windows NT, da es sich bei Windows NT um ein stabiles Betriebssystem mit exzellenter Prozessisolation handelt, eine Notwendigkeit für die Arbeit auf API-Ebene –, und Windows 95 und Windows 98 sind ... nun ja, es sind Betriebssysteme. Windows 2000 wird im Hinblick auf die in diesem Buch angeführten Anwendungen wahrscheinlich ein ähnliches Verhalten wie NT 4.0 aufweisen – vorausgesetzt natürlich, es wird jemals ausgeliefert.

Unterschiede zu Windows 95/98 werden nur dann aufgeführt, wenn sich diese auf die Ergebnisse beziehen, da ein professioneller Entwickler nicht auf die Systemunterschiede hingewiesen werden muss, die sich bei der Anwendungsentwicklung ergeben. Bei einer Durchsicht der Puzzle wird Ihnen auffallen, dass diese Art der Entwicklung immer unter Windows NT durchgeführt werden sollte, da unter NT gegenüber Windows 95/98 eine bessere Fehlerermittlung sowie detailliertere Fehlerberichte verfügbar sind.

Was Sie in diesem Buch nicht finden

In diesem Buch wird der Aufruf von API- und DLL-Funktionen von Visual Basic aus behandelt.

Wenn Sie noch ein Anfänger sind, fragen Sie sich vielleicht »Was ist eine API-Funktion?« oder »Was ist eine DLL-Funktion?«. Vielleicht machen Sie sich auch darüber Sorgen, dass Sie nicht wissen, wie Windows funktioniert. Sie kennen unter Umständen die Begriffe **Fensterzugriffsnummer** oder **Gerätekontext** nicht.

Tja, ich werde Sie Ihnen nicht erklären.

Dieses Buch behandelt nicht die Win32-API selbst. Dieses Buch habe ich bereits geschrieben. Es heißt **Dan Appleman's Visual Basic Programmer's Guide to the Win32 API**. Dieses Buch ist ein Bestseller unter den Büchern für Visual Basic-Programmierer und kann in jedem Buchhandel mit gut sortierter Computerliteratur erworben werden (oder Sie bestellen es direkt bei Desaware, Inc. unter **http://www.desaware.com/** oder bei einer anderen Mailorderfirma). Die ersten vier Kapitel dieses Buches enthalten Informationen, mit denen selbst ungeübte VB-Programmierer die Grundlagen der API-Programmierung von Visual Basic aus erlernen können. Der verbleibende Teil des Buches beschäftigt sich mit dem Kern der Win32-API, der Funktionsweise von Windows sowie der API-Funktionen. Das Buch umfasst 1500 Seiten (plus einige weitere Kapitel auf CD-ROM). Wenn Sie denken, dass ich in der Lage wäre, die Informationen dieses Buches in diesem viel weniger umfangreichen Buch zusammenzufassen, haben Sie sich leider getäuscht. Und nicht nur das, warum sollte ich mir deswegen Sorgen machen?

Schließlich handelt es sich bei diesem Puzzlebuch nicht um eine umfassende Referenz. In diesem Buch soll Ihnen nicht die Funktionsweise der Win32-API beigebracht werden. Statt dessen lernen Sie, wie Sie die C/C++-Dokumentation interpretieren, um eigene Deklarationen zu erstellen und komplexe API-Probleme selbst zu lösen. Dieses Buch soll Ihnen dabei helfen, die Mechanismen eines API-Aufrufs zu verstehen, damit Sie die Informationen in meinem API-Buch effektiv anwenden können, auch wenn keine Beispiele vorhanden sind, oder damit Sie eine Funktion auf eine Weise anwenden können, die in keinem der Beispiele erläutert wird.

Warum dies? Weil die Win32-API ständig größer wird – es werden täglich neue Funktionen hinzugefügt – und ich kann meine Zeit nicht darauf verwenden, Mammutbücher zu schreiben, in denen all diese Funktionen beschrieben werden. Darüber hinaus scheint sich außer mir keiner für diesen Job zu interessieren.

Wenn Sie nicht in der Lage sind, sämtliche Puzzle in diesem Buch zu lösen, heißt dies nicht, dass Sie ein weiteres dieser Mammutbücher zur Win32-API lesen müssen. Sie werden fähig ein, die C/C++-Dokumentation zu verwenden und schnell und genau eigene Deklarationen zu erstellen. Nicht nur dies, Sie werden qualifiziert sein, Ihr eigenes VB-Programmiererhandbuch zu einer beliebigen der Win32-Erweiterungsbibliotheken zu schreiben.

Und nach all der Werbung bleibt noch zu sagen, dass mir ein paar Zeilen ihrerseits jederzeit willkommen sind – wenn Sie denn wirklich wollen.

Die zehn Gebote der sicheren API-Programmierung (überarbeitete Fassung)

Ich habe die ursprünglichen »Zehn Gebote der sicheren API-Programmierung« um 1992 geschrieben, und seither sind verschiedene Versionen dieser Gebote in Publikationen, Konferenzen, Websites und Büchern erschienen. Ich bin der Meinung, dass durch die Einhaltung dieser Gebote nahezu jedes Problem bei der API-Programmierung gelöst werden kann, aus mir nicht bekannten Gründen scheinen einige Leute dies jedoch nicht ohne weiteres glauben zu wollen. Vielleicht kann ich diese Leute mit dem vorliegenden Buch davon überzeugen, da viele der Puzzle Hinweise enthalten, die auf eines der zehn Gebote verweisen.

Machen Sie sich keine Sorgen, wenn Ihnen einige (oder alle) Konzepte in den Geboten im Moment unklar erscheinen. Wenn Sie sich durch die Puzzle, Lösungen und Tutorien arbeiten, werden sie Ihnen schnell in Fleisch und Blut übergehen.

1 Gedenke der ByVal-Anweisung und halte Sie in Ehren!

Die richtige Verwendung der `ByVal`-Anweisung ist der wichtigste Faktor bei der erfolgreichen API-Programmierung über Visual Basic. Denken Sie daran, dass das `ByVal`-Schlüsselwort manchmal nicht nur in der Deklaration, sondern im Funktionsaufruf selbst verwendet wird. Zur Anwendung weitergehender Techniken ist es zwingend erforderlich, die Funktionsweise von `ByVal` genau zu kennen und zu wissen, welche Auswirkungen diese Anweisung auf den API-Funktionsaufruf von Visual Basic aus hat.

2 Du sollst deine Parametertypen überprüfen!

Das Verwenden der richtigen Parametertypen ist unter Win32 noch wichtiger geworden als unter Win16. Unter Win32 werden die meisten Parameter, unabhängig von ihrer tatsächlichen Größe, als 32-Bit-Werte übergeben. Daher werden über die traditionelle Fehlermeldung »bad DLL calling convention« weniger Fehler ermittelt als unter Win16. Das Ergebnis sind gut versteckte, datenabhängige Bugs, die nur sehr schwer ausfindig zu machen sind. Denken Sie stets daran, dass bei vielen API-Funktionen für einen Parameter mehrere Datentypen verwendet werden können, was es überaus wichtig macht, die Unterschiede bei der Übergabe der verschiedenen Parametertypen an die Funktionen genau zu kennen. Denken Sie außerdem daran, dass Visual Basic eine tückische Konvertierungseigenschaft besitzt, durch die ohne Vorwarnung eine automatische Wertetypenumwandlung vorgenommen wird. Dies kann dazu führen, dass Sie einen Ihrer Meinung nach richtigen Parametertyp übergeben, der tatsächlich jedoch einen ungültigen Wert enthält.

3 Du sollst deine Rückgabetypen überprüfen!

Durch die meisten API-Funktionen werden 32-Bit-Werte zurückgegeben. Probleme bei der Werterückgabe treten häufig dann auf, wenn von einer Deklaration kein Wert vom Typ `Long` zurückgegeben wird. Der häufigste Fehler hierbei besteht darin, dass für die Funktionsdeklaration kein Rückgabetyp angegeben wurde. In diesem Fall wird von Visual Basic der Standardrückgabetyp `Variant` verwendet, der mit Sicherheit falsch ist und den Fehler »bad DLL calling convention« oder einen Speicherausnahmefehler auslöst.

4 Du sollst deine Zeichenfolgen initialisieren, sonst wirst du untergehen!

API-Funktionen, die Zeichenfolgenparameter verwenden, erkennen diese üblicherweise als Adressen zu Speicherstellen, die eine auf einen Nullwert endende Zeichenfolge enthalten. Viele Funktionen können Zeichenfolgenwerte wieder an die aufrufende VB-Anwendung zurückgeben, indem diese Speicherstelle mit den Daten geladen wird. Die API-Funktion kann jedoch nicht ermitteln, wieviel Platz im verfügbaren Puffer vorhanden ist. Bei einigen Funktionen kann die Puffergröße als separater Parameter übergeben werden. Bei anderen Funktionen dagegen ist einfach eine bestimmte Puffergröße erforderlich. Wenn Sie eine noch nicht initialisierte oder einen leere Zeichenfolge an eine Funktion übergeben, weist der Speicherpuffer entweder keinen Platz oder einen einzelnen Zeichenwert auf (der auf NULL endende Zeichenwert). Wenn die API-Funktion versucht, Daten an die bereitgestellte Speicheradresse zu laden, werden mit Sicherheit wichtige Daten in Ihrem Anwendungsspeicher überschrieben. Dies kann zu sofortigen Speicherausnahmefehlern bis hin zu schwer auffindbaren Bugs führen, die nur dann auftreten, sobald Sie bereits Tausende von Kopien Ihrer Anwendung ausgeliefert haben. Stellen Sie deshalb immer sicher, dass Sie die Zeichenfolgenpuffer initialisieren, falls auch nur im Entferntesten die Möglichkeit besteht, dass der Puffer durch die API-Funktion bearbeitet wird.

5 Verwende nicht As Any, denn es ist böse!

Wenn Sie einen Parametertyp als `As Any` deklarieren, sind Sie allein dafür verantwortlich, den richtigen Datentyp an den Code zu übergeben, durch den die Funktion aufgerufen wird. In Visual Basic wird keinerlei Typenprüfung vorgenommen. Die `As Any`-Deklarationen werden am häufigsten bei API-Funktionsparametern verwendet, die mehr als einen Datentyp verwenden können. Glücklicherweise können dank der Option `Alias` in der `Declare`-Anweisung in Visual Basic mehrere Deklarationen für dieselbe Funktion erstellt werden.

Dennoch gibt es Fälle, in denen es sehr viel bequemer ist, den Parametertyp `As Any` beizubehalten. So kann beispielsweise eine Funktion so viele verschiedene Datentypen verwenden, dass es zu umständlich wäre, für jeden Datentyp eine andere Deklaration zu definieren – und sich diese zu merken. Oder die Funktion verfügt über mehr als einen Parameter, der verschiedene Datentypen annehmen kann, wodurch für jeden neuen Typ mehrere Deklarationen erstellt werden müssen.

Ganz gleich, welchen Ansatz Sie wählen, es ist äußerst wichtig, den `As Any`-Parameter und dessen Funktion zu verstehen. Außerdem kann man nie wissen, wann man vielleicht zufällig an den Code eines anderen Entwicklers gerät, der diese Regeln leider nicht eingehalten hat.

6 Du sollst Option Explicit angeben!

Bei der Arbeit mit Visual Basic sollte immer die Editoroption **Variablendeklaration erforderlich** gesetzt werden. Wenn Sie Code überarbeiten, der erstellt wurde, ohne dass diese Option gesetzt wurde, sollten Sie sicherstellen, dass Sie den Befehl **Option Explicit** zu Beginn eines jeden Codemoduls hinzufügen (einschließlich **Formular, Benutzersteuerelement, Klassenmodul** usw.).

Setzen Sie diese Option nicht, wird durch jeden Verweis auf eine Variable automatisch eine leere Instanz dieser Variablen erstellt. Hierzu gehören auch Fälle, in denen versehentlich eine neue Variable erstellt wird, da der Name einer vorhandenen Variablen falsch geschrieben wurde. Schlimmer noch ist, aus der Sicht eines API-Programmiers, dass die neu erstellte Variable sehr wahrscheinlich den Typ `Variant` erhält. Unabhängig von der Sichtweise entsprechen die Ergebnisse jedoch höchstwahrscheinlich nicht denen, die Sie erzielen wollten. Der zusätzliche Aufwand, der durch die Deklaration jeder Variablen vor deren Verwendung entsteht, ist jedoch nichts im Vergleich zu der mühseligen Arbeit, die bei der Suche nach versteckten Bugs entsteht, die durch typografische Fehler verursacht werden.

7 Ehre deine VB- und Win32-Ganzzahlen, denn sie stimmen nicht überein!

Eine Ganzzahl umfasst 16 Bits. Oder nicht? Wenn man in der C- oder C++-Dokumentation nachschlägt (der Standardsprache für die gesamte Windows-Dokumentation), beziehen sich unter Win32 alle `Int`-Werte und sämtliche Verweise auf Ganzzahlen auf einen 32-Bit-Wert. Wenn eine Person sich also auf »`Integer`-Werte« (Ganzzahlen) bezieht oder auf diese in einem Buch verwiesen wird, sollte man immer den jeweiligen Kontext berücksichtigen.

8 Überprüfe stets deine Funktionsnamen, denn es wird nun die Groß- und Kleinschreibung berücksichtigt, und die Funktionsnamen können Suffixe aufweisen!

Unter Win32 wird bei sämtlichen API-Funktionen die Groß-/Kleinschreibung beachtet. Im Gegensatz dazu spielte die Groß- oder Kleinschreibung der Funktionsnamen bei der Win16-API keine Rolle. Achten Sie in Situationen, in denen die Funktion einen Zeichenfolgenwert verwendet, außerdem auf Suffixe. Es kann oft

vorkommen, dass der Name der Funktion in der DLL über die angehängten Buchstaben »A« oder »W« verfügt, durch den der ANSI- oder Unicode-Einsprungpunkt der Funktion angegeben wird.

9 Du sollst deine Parameter und Rückgabewerte prüfen!

Eine der schönsten Eigenschaften von Visual Basic ist die, dass es sich um eine interpretierte Sprache handelt. Dies bedeutet, dass Sie Ihren Code bei der Ausführung an einem beliebigen Punkt anhalten können, um die Werte der einzelnen Parameter zu überprüfen. Wenn ein API-Aufruf nicht wie erwartet funktioniert, stoppen Sie an dem Punkt und untersuchen die im Aufruf verwendeten Parameterwerte. Suchen Sie nach Werten, die keinen Sinn ergeben, und achten Sie dabei besonders auf Parameter, die gültige Daten enthalten sollten, statt dessen jedoch den Wert 0 aufweisen (Hinweis auf einen Fehler in einem vorangegangenen API-Aufruf). Überprüfen Sie die Rückgabewerte der Funktionen, und rufen Sie mit Hilfe der Methode `Err.LastDllError` zusätzliche Fehlerinformationen ab.

10 Speichere deine Arbeit so oft wie möglich!

Das Gute an der Verwendung von API-Funktionsaufrufen ist, dass diese nach der richtigen Deklaration und Programmierung extrem zuverlässig sind. Weniger gut bei der Verwendung von API-Funktionen ist es, dass, bis Sie diese richtig deklariert und programmiert haben, die erzeugten Fehler häufig zu einer Speicherausnahme und somit zu einem Programmabsturz, zu einem Hängenbleiben von Visual Basic und in einigen Fällen sogar zu einem Systemabsturz führen. Nichts ist frustrierender als nach dem Hinzufügen dutzender neuer Codezeilen auf die Schaltfläche **Starten** zu klicken, nur um mitzuerleben, wie dieser Code durch einen Speicherausnahmefehler im Nichts verschwindet. Zu meinem eigenen Besten speichere ich deshalb meinen Code, bevor ich ihn ausführe. Und das möchte ich auch Ihnen dringend empfehlen.

Teil I
Die Puzzle

Es gibt verschiedene Möglichkeiten, eine Reise zu planen. Manche Menschen treffen Vorbereitungen, packen sorgfältig ihre Koffer, lernen ein paar Sätze in der jeweiligen Sprache des Urlaubslandes und planen jedes Detail der Reise so genau wie möglich. Andere werfen lediglich eine Unterhose und eine Zahnbürste in ihren Rucksack, besorgen sich das nächstbeste Flugticket und los geht's ...

Man kann jedes Buch als eine Art Reise betrachten. Vergessen wir hierbei für einen Moment die schlechten Bücher, die Reisen ähneln, die Sie bei einem fragwürdigen Reiseveranstalter gebucht haben, der sich nach Antritt der Reise aus jeder Verantwortung stiehlt und Sie allein im tiefsten Dschungel zurücklässt, wo Sie plötzlich einem Löwen gegenüberstehen, der Sie ansieht, als sei Ihr Name »Abendessen«.

Selbst die guten Bücher (zu denen dieses hoffentlich zählt) können sich vom Stil her stark unterscheiden. Mit einigen Büchern reisen Sie in der ersten Klasse. Diese Bücher sind freundlich aufgemacht und bereiten Sie in einer luxuriösen Umgebung auf jeden Ihrer nächsten Reiseschritte vor.

Das vorliegende Buch ähnelt eher einer Abenteuerreise. Es soll für Sie eine Herausforderung darstellen und selbst die erfahrensten Programmierer ins Schlingern bringen. Dem liegt die Idee zugrunde, Ihr Wissen und Ihre Erfahrung zu erweitern, damit Sie Vertrauen in Ihre eigenen Fähigkeiten erlangen und jede Herausforderung annehmen, die sich Ihnen entgegenstellt.

Wenn Sie eher der Typ Reisender sind, der vorausplant, sollten Sie zu Teil III wechseln und die Tutorien lesen, um sich mit den Zielen vertraut zu machen, zu denen Sie demnächst vorstoßen werden.

Aber wenn Sie zu dem Typ Reisenden gehören, der sich einfach in das nächste Abenteuer stürzt, dann lesen Sie einfach weiter. Die Tutorien können Sie immer noch lesen, wenn (falls) Sie auf Probleme stoßen.

Abschnitt 1: Der Anfang

Haben Sie jemals ein Programmierhandbuch für Fortgeschrittene gekauft? Dann haben Sie sich bestimmt auch gefragt, warum das erste Kapitel mit einer ausführlichen Erläuterung von Variablen, einfachen Operationen oder Aufgaben wie z. B. dem Hinzufügen von Steuerelementen oder Formularen beginnen muss. Ich fand diese Art von Büchern schon immer sehr frustrierend. Nicht, weil etwas dagegen spräche, diese Art von Informationen zur Verfügung zu stellen, sondern weil sich die weiterführenden Informationen, nach denen ich üblicherweise suche, auf einige wenige Kapitel am Ende des Buches beschränken, wobei diese dann häufig auch noch wenig hilfreich sind.

Auf der anderen Seite müssen Sie jedoch auch das Dilemma verstehen, in dem sich jeder Autor befindet: Wenn Sie ein Buch auf einer zu weit fortgeschrittenen Ebene beginnen, riskieren Sie, eine Menge Leser zu verlieren und weitaus weniger Bücher zu verkaufen[1]. Wenn das Buch auf einer zu einfachen Ebene beginnt, frustrieren Sie den professionellen Entwickler, der es hasst, Aufgüsse von dem zu lesen, was er längst weiß. Es handelt sich also um eine Gratwanderung, die es zu absolvieren gilt.

Daher richtet sich dieses Buch an Visual Basic-Programmierer mit fortgeschrittenen Kenntnissen, die sich zu Experten mausern möchten. Die Puzzle setzen voraus, dass der Leser über eine gewisse Erfahrung im Umgang mit API-Funktionsaufrufen verfügt. Alle Leser, die diese Kenntnisse nicht besitzen, sollten unbedingt die Tutorien lesen, bevor Sie mit der Bearbeitung der Puzzle beginnen. Aufgrund dieser Annahmen habe ich mir die Freiheit genommen, auch die ersten Puzzle in diesem Abschnitt etwas schwieriger zu gestalten[2], als man vielleicht bei einem ersten Kapitel erwarten würde. Ich denke, selbst der erfahrene Programmierer wird bei diesen Puzzlen noch einmal genauer hinsehen müssen, auch wenn sich die Lösung im Nachhinein als sehr einfach herausstellt.

1. Ich weiß, dass dies ein bisschen kapitalistisch klingt, aber ich denke, es ist wichtig, dass die Leser ein wenig über die wirtschaftliche Seite der Verlagsbranche erfahren sollten, denn erst dadurch wird die Veröffentlichung vieler lausiger Bücher erklärbar. Ich habe vor kurzem ein Essay mit dem Titel »Are you learning Visual Basic backwards?« geschrieben, den Sie sich auf unserer Website unter **www.desaware.com** ansehen können.
2. OK, vielleicht in einigen Fällen auch noch etwas schwieriger.

Puzzle 1

Wo steckt denn jetzt der API-Aufruf?

Wir leben in einer kleinen Welt, und Sie können nie wissen, wann Sie Ihre Software anpassen müssen, sodass Sie auch in anderen Teilen dieser Welt ausgeführt werden kann. Obwohl es vielleicht merkwürdig erscheint, bestehen die anderen Länder weiterhin darauf, eigene Sprachen, Währungs- und Satzzeichen zu verwenden. Nicht nur das, sie verwenden auch noch andere Datums- und Zeitformate.[3]

Windows verwendet den Begriff »Ländereinstellung«, um einen Standort und dessen spezifische Merkmale zu definieren. Jede Ländereinstellung verfügt über eine eindeutige Nummer. Glücklicherweise ist es in Windows sehr einfach, die Merkmale der Ländereinstellung zu ermitteln, unter der eine Anwendung ausgeführt wird.

Nachfolgend ein einfaches Programm, mit dem Sie die Nummer der aktuellen Ländereinstellung ermitteln können:

```
Private Declare Function GetUserDefaultLcid Lib "User32" () As Long
Private Sub Command1_Click()
   Dim lcid&
   Dim info$
   lcid = GetUserDefaultLcid()
   MsgBox lcid, vbOKOnly, "User Default LCID"
End Sub
```

Ergebnisse

Laufzeitfehler 453:

```
DLL-Einsprungspunkt GetUserDefaultLcid in User32 nicht gefunden
```

Wo steckt denn nun der API-Aufruf? Berichtigen Sie die Deklaration, damit das Programm funktioniert.

3. Obwohl mir häufig vorgeworfen wird, dass ich sehr USA-bezogen eingestellt sei, möchte ich doch betonen, dass der vorangegangene Absatz sich auf jede Sprache beziehen lässt, in der Sie entwickeln.

Puzzle 2

Der letzte Fehler

Eine faszinierende Aufgabe, der sich Windows-Programmierer gelegentlich gegenübersehen, ist die Verwendung von Programmen zur Steuerung des Systems oder einer anderen Anwendung. Der erste Schritt bei solchen Programmen besteht darin, die Fensterzugriffsnummer für die andere Anwendung herauszufinden. Die API-Funktion `FindWindow` kann, bei vorhandenem Titel bzw. vorhandener Klasse, dazu verwendet werden, diese Fensterzugriffsnummer zu ermitteln. Die C-Deklaration für `FindWindow` lautet folgendermaßen:

```
HWND FindWindow(
   LPCTSTR lpClassName, // Zeiger auf den Klassennamen
   LPCTSTR lpWindowName // Zeiger auf den Fensternamen
);
```

Diese Funktion wurde so konzipiert, dass entweder die Fensterklasse (der Fenstertyp) oder deren Name (der Titel) angegeben werden kann. Sie können auch beides angeben, was allerdings nicht notwendig ist. Da im vorliegenden Beispiel der Fenstertitel verwendet werden soll, setzen Sie den Klassennamen auf NULL.

Sie könnten die Deklaration für `FindWindow` in der Datei **api32.txt** nachsehen (diese befindet sich auf der Begleit-CD-ROM zu diesem Buch), oder Sie schauen in die Datei **win32api.txt**, die im Lieferumfang von Visual Basic enthalten ist, aber warum versuchen Sie nicht, sie selbst herauszufinden?

Da `FindWindow` Bestandteil des Fenstersubsystems ist, wird `User32` verwendet (`kernel32` enthält OS-Kernfunktionen, `GDI32` enthält Grafikfunktionen; Beschreibungen hierzu finden Sie in Teil III dieses Buches unter Tutorium 1, »Auffinden von Funktionen«). Da diese Funktion Zeichenfolgenparameter verwendet, wissen Sie, dass sowohl ANSI- als auch Unicode-Einsprungpunkte vorhanden sind, d.h., Sie müssen einen Alias verwenden, um auf den ANSI-Einsprungpunkt zugreifen zu können.[4] Ein HWND ist eine Zugriffsnummer, die unter Win32 den Wertetyp `Long` aufweist, folglich ist der Rückgabewert vom Typ `Long`. Beide Parameter sind Zeichenfolgen, die `ByVal` übergeben werden müssen, damit Sie als auf NULL endende Zeichenfolgen übertragen werden. Die Deklaration lautet also:

4. Wenn Sie nicht wissen, dass API-Funktionen mit Zeichenfolgenparametern separate Einsprungpunkte aufweisen, oder keine Ahnung haben, was mit den Begriffen »ANSI«, »Unicode« oder »Einsprungpunkt« gemeint ist, dann sollten Sie dieses Puzzle unterbrechen und Tutorium 5, »Das `ByVal`-Schlüsselwort – die Lösung für 90 % aller API-Probleme«, in Teil III dieses Buches lesen.

```
Declare Function FindWindow Lib "User32" Alias "FindWindowA" _
(ByVal lpClassName As String, ByVal lpWindowName As Long) As Long
```

Das Projekt **LastErr.vbp** (auf der Begleit-CD-ROM) beinhaltet ein Formular mit einem Textfeld `txtCaption`. In dieses können Sie den Titel des Hauptfensters für ein beliebiges Programm eingeben. Über die Funktion `FindWindow` wird die Zugriffsnummer für dieses Fenster abgerufen. Anschließend verwendet das Programm die Funktion `PostMessage`, um eine WM_CLOSE-Nachricht in die Nachrichtenwarteschlange für das Fenster zu stellen – die gleiche Nachricht, die gesendet wird, wenn Sie das Feld **Schließen** oder das Systemmenü zum Schließen eines Fensters verwenden.

```
Private Declare Function PostMessage Lib "User32" Alias _
"PostMessageA" (ByVal hWnd As Long, ByVal Message As Long, ByVal _
wParam As Long, ByVal lParam As Long) As Long
Private Const WM_CLOSE = &H10
Private Sub cmdClose_Click()
   Dim WindowHandle As Long
   WindowHandle = FindWindow("", txtCaption.Text)
   If WindowHandle = 0 Then
     MsgBox "No window: Last Error was " & Err.LastDllError
     Exit Sub
   Else
     Call PostMessage(WindowHandle, WM_CLOSE, 0, 0)
   End If
End Sub
```

Ergebnisse

Die standardmäßige Bezeichnung lautet **Untitled – Notepad**, dem Namen des Anwendungsfensters eines Standardeditors. Das Programm schlägt fehl. Im Meldungsfeld wird der Fehlercode `LastError` angezeigt, ein Wert, der vom Betriebsystem ausgegeben wird und über den Fehlerinformationen bereitgestellt werden.[5] Ihre Aufgabe besteht aus zwei Arbeitsschritten:

1. Finden Sie die Bedeutung des letzten Fehlercodes heraus.
2. Ändern Sie das Programm so ab, dass es funktioniert.

5. Der Fehlercode `LastDllError` trägt unter Windows NT die Nummer 123, unter Windows 95/98 lautet die Fehlernummer 0.

Puzzle 3

Möchte Poly einen Keks?

Die GDI-Schnittstelle (Graphical Device Interface) von Windows verfügt über weitaus mehr Grafikfunktionen als in Visual Basic zur Verfügung stehen. Eine dieser Funktionen erlaubt Ihnen das Zeichnen eines Polygons mit beliebig vielen Seiten in nur einem Arbeitsschritt. Diese Polygon-Funktion wird in der C-Dokumentation folgendermaßen definiert:

```
BOOL Polygon(
HDC hdc                 // Zugriffsnummer für den Gerätekontext
CONST POINT *lpPoints,  // Zeiger auf die Scheitelpunkte des Polygons
int nCount              // Zählung der Scheitelpunkte des Polygons
);
```

Der `hdc`-Parameter stellt den Gerätekontext für das Fenster dar, in dem das Polygon gezeichnet wird. Bei dem `lpPoints`-Parameter handelt es sich um einen Zeiger auf die erste von mehreren Koordinaten, mit denen die zu verbindenden Punkte beschrieben werden. Der Parameter `nCount` bezeichnet die Anzahl der Punkte.

Können Sie das folgende Programm so abändern, dass die Figur in Abbildung P3-1 angezeigt wird?

```
' Poly Example Program
' Copyright © 1998 by Desaware Inc. Alle Rechte vorbehalten
Option Explicit
Private Type POINTAPI
   X As Long
   Y As Long
End Type
Private Declare Function Polygon Lib "gdi32" (hdc As Long, _
lpPoint As POINTAPI, nCount As Long) As Long
' Funktion lädt ein Array von Punkten
Private Sub LoadPointArray(ByVal Width As Long, ByVal Height _
As Long, ByVal Increment As Integer, PointArray() As POINTAPI)
   Dim curidx As Integer
   ReDim PointArray((Height \  Increment) + 2)
   Do
     curidx = curidx + 1
     PointArray(curidx).X = Width
     PointArray(curidx).Y = Height - curidx * Increment
```

```
      curidx = curidx + 1
      PointArray(curidx).X = 0
      PointArray(curidx).Y = curidx * Increment
   Loop While curidx * Increment < Height

End Sub

Private Sub Form_Paint()
   Dim points() As POINTAPI
   LoadPointArray Width, Height, 5, points()
   Call Polygon(hWnd, points(0), UBound(points) + 1)
End Sub

Private Sub Form_Resize()
   Refresh
End Sub
```

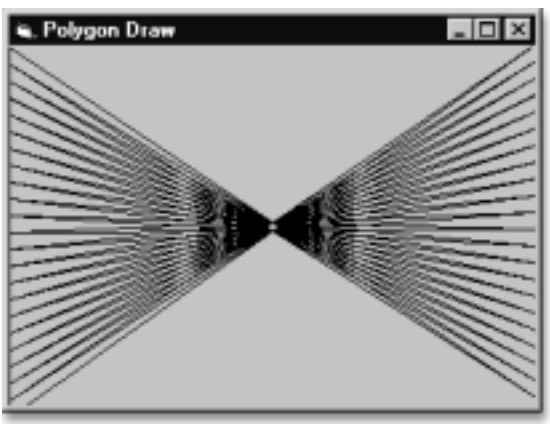

Abbildung P3-1 Richtige Anzeige für das Poly-Programm

Ergebnisse

Nach der Ausführung des oben aufgeführten Codes wird ein leeres Formular angezeigt.

Können Sie den Code berichtigen? Die Aufgabe kann schwieriger sein, als sie zunächst aussieht.

Puzzle 4

Nomen est Omen

Wir geben unseren Computern gerne Namen. Nein, diese Tatsache rührt nicht daher, dass wir einem Computer etwas menschlichere Züge verleihen möchten. Es ist halt so, dass bei der Installation von Windows ein Computername angegeben werden muss.

Glücklicherweise ist es mit Hilfe der Win32-API sehr einfach, über ein Programm den Namen des Computers zu ermitteln, auf dem dieses Programm ausgeführt wird.

In C wird die Funktion `GetComputerName` folgendermaßen deklariert:

```
BOOL GetComputerName(
LPTSTR lpBuffer,   // Adresse des Namenspuffers
LPDWORD nSize      // Adresse für die Größe des Namenspuffers
);
```

Sie könnten sich das folgende einfache Programm ausdenken, mit dem der Computername abgerufen und anschließend in einem Namensfeld angezeigt wird:

```
' Computer Name
' Copyright © 1998 by Desaware Inc. Alle Rechte vorbehalten

Option Explicit

Private Declare Function GetComputerName Lib "kernel32" (ByVal _
ComputerName As String, ByVal BufferSize As Long) As Long

Private Const MAX_COMPUTERNAME_LENGTH = 15

Private Sub Form_Load()
   Dim s$
   Call GetComputerName(s$, MAX_COMPUTERNAME_LENGTH + 1)
   lblName.Caption = s$
End Sub
```

Ergebnisse

Das Ergebnis mag als Kommentar zu einem Code genügen, den Sie sich einfach ausdenken. Wie wäre es, wenn Sie nun Hand anlegen und das Programm so abändern, dass es funktioniert?

Puzzle 5

Finden Sie den Namen des ausführenden Programms!

Sie kennen den Namen Ihres Programms. Schließlich haben Sie es kompiliert und sehr wahrscheinlich auch installiert. Wenn Sie jedoch Softwarekomponenten erstellen, können diese Komponenten von vielen anderen Anwendungen aufgerufen werden, und Sie wissen nicht unbedingt, welche Anwendung Ihre Komponente verwendet.

Warum sollten Sie dies wissen wollen? Der einfachste Grund dafür wäre, dass Sie die Komponente lizenzieren möchten. Sie möchten in diesem Fall ermitteln können, ob die Komponente innerhalb der Visual Basic-Laufzeitumgebung oder innerhalb eines kompilierten ausführbaren Programms verwendet wird. Wird die Komponente in der VB-Umgebung ausgeführt, können Sie überprüfen, ob eine Lizenz vorliegt, bevor Sie die Komponente zur Ausführung freigeben.

Über die Win32-API ist es sehr einfach, den Namen des Programms zu ermitteln, durch den ein Prozess ursprünglich gestartet wurde. Dies gelingt mit der Funktion `GetModuleFileName`. Diese Funktion wird im Win32 Software Development Kit (SDK) folgendermaßen definiert:

```
DWORD GetModuleFileName(
   HMODULE hModule,   // Zugriffsnummer des Moduls, nach dessen Dateiname
                      // gesucht wird
   LPTSTR lpFilename, // Zeiger auf Puffer, der den Modulpfad empfangen
                      // soll
   DWORD nSize        // Puffergröße, in Zeichenwerten
   );
```

Die Konvertierung dieser Parametertypen in Visual Basic sollte nicht schwer fallen. Bei Parameter `hModule` handelt es sich um eine Zugriffsnummer, daher muss es sich um einen 32-Bit-Wert vom Typ `Long` handeln. Der `lpFilename`-Parameter ist eine Zeichenfolge, die mit dem vollständigen Pfad sowie dem Namen der ausführbaren Datei geladen wird. Dieser Parameter muss `ByVal` als Zeichenfolge deklariert werden. Bei dem Parameter `nSize` handelt es sich um einen weiteren 32-Bit-Wert des Typs `Long`, der die Länge des Zeichenfolgenpuffers enthält. Und da die Funktion im Rahmen der Prozessverwaltung eingesetzt wird, befindet sie sich in der DLL (Dynamic Link Library) `kernel32`. Die Funktion kann also folgendermaßen deklariert werden:

```
Private Declare Function GetModuleFileName Lib "kernel32" Alias _
"GetModuleFileNameA" (ByVal hModule As Long, ByVal lpFileName As _
String, ByVal nSize As Long) As Long
```

Der Parameter `hModule` bedarf einer weiteren Erläuterung. Eine Modulzugriffsnummer unter Windows enthält die Adresse, an der das angegebene Modul in den Speicher geladen wird. Beim Start einer Anwendung wird die ausführbare Datei im Speicher zugeordnet, beginnend mit der Moduladresse für die ausführbare Datei. Jede von der Anwendung verwendete DLL wird einer anderen Adresse zugeordnet, die zur Moduladresse für diese DLL wird.[6] Mit der Funktion `GetModuleFileName` kann der Dateiname für ein beliebiges von einem Prozess verwendetes Modul abgerufen werden.

Im vorliegenden Fall wenden wir einen Trick an: Wenn Sie den Parameter `hModule` auf 0 setzen, wird über die Funktion der Name der Anwendung abgerufen, die ursprünglich den Prozess gestartet hat.

Das Projekt `ExecutableFinder` ist eine einfache ActiveX-DLL-Komponente, die ein einklassiges Objekt mit dem Namen `Server` enthält. Dieses Objekt stellt eine einzige Methode bereit, mit der das aufrufende Element ermitteln kann, ob es sich bei der Art der ausgeführten Anwendung um Visual Basic handelt. Wenn über eine Anwendung ein Serverobjekt erstellt wird, wird die `ExecutableFinder`-DLL in den Prozess geladen. Die `IsThisVB`-Methode verwendet die `GetModuleFileName`-Funktion, um zu ermitteln, ob es sich bei dem ausgeführten Programm um ein Visual Basic-Programm handelt. Wenn Sie ein Programm innerhalb der Visual Basic-Entwicklungsumgebung ausführen, gibt `GetModuleFileName` an, dass es sich, je nachdem, ob Sie VB6, VB5 oder VB4 verwenden, um das Programm **vb6.exe**, **vb5.exe** oder **vb4.exe** handelt. Wenn Sie ein kompiliertes ausführbares Programm aufrufen, wird der Pfad dieses Programms abgerufen.

Die `Server`-Klasse enthält den folgenden Code:

```
' What's the Executable?
' Copyright © 1998 by Desaware Inc. Alle Rechte vorbehalten
Option Explicit

Private Declare Function GetModuleFileName Lib "kernel32" Alias _
"GetModuleFileNameA" (ByVal hModule As Long, ByVal lpFileName As _
String, ByVal nSize As Long) As Long
Private Const MAX_PATH = 260
```

6. In Anhang B, »Häufig gestellte Fragen«, finden Sie weitere Informationen zu Modulzugriffsnummern.

```
Public Function IsThisVB() As Boolean
   Dim ExecName As String
   Dim LastBackslashPos As Long
   ExecName = String$(MAX_PATH + 1, 0)
   Call GetModuleFileName(0, ExecName, MAX_PATH)
   ' Jetzt den Pfad abschneiden
   LastBackslashPos = InStrRev(ExecName, "\ ")
   If LastBackslashPos = 0 Then
     LastBackslashPos = InStrRev(ExecName, ":")
   End If
   ExecName = Mid$(ExecName, LastBackslashPos + 1)
   If LCase$(ExecName) = "vb6.exe" Or _
     LCase$(ExecName) = "vb5.exe" Or _
     LCase$(ExecName) = "vb32.exe" Then
     IsThisVB = True
   End If

End Function
```

Der Parameter `ExecName` ist eine Zeichenfolge, die durch MAX_PATH + 1 Bytes initialisiert wird. Dieser Wert ist lang genug, um den längsten vom System unterstützten Pfad aufzunehmen. Die `GetModuleFileName`-Funktion lädt die Zeichenfolge mit dem vollständigen Pfad der ausführbaren Datei, durch die der Prozess gestartet wurde.

In diesem Fall ist der vollständige Pfad unwichtig – wichtig ist nur der zuletzt aufgeführte Name der ausführbaren Datei. Sie können den Pfad entfernen, indem Sie nach dem letzten Backslash suchen (oder nach dem Doppelpunkt, falls das Programm vom Stammverzeichnis aus ausgeführt wird). Mit Hilfe der Funktion `Mid$` werden alle Zeichen vor dem Backslash aus der Zeichenfolge entfernt. Abschließend wird unter Berücksichtigung der Groß- und Kleinschreibung ein Vergleich mit den drei Namen der Visual Basic-Programme durchgeführt. Die Funktion gibt den Wert `True` zurück, wenn eine Übereinstimmung mit einem der drei Programmnamen konstatiert wird.

Ergebnisse

Fügen Sie der Visual Basic-Umgebung zu Testzwecken ein einfaches Projekt hinzu, wie in der `IsThisVB`-Gruppe gezeigt.

```
' IsThisVBTest Sample Program
' Copyright © 1998 by Desaware Inc. Alle Rechte vorbehalten
```

```
Option Explicit

Private Sub Form_Load()
   Dim serverobject As New Server
   If serverobject.IsThisVB Then
     Label1.Caption = "Yes - it's VB"
   Else
     Label1.Caption = "No - it's not VB"
   End If
End Sub
```

Aus irgendeinem Grund kann mittels dieses Programms nicht ermittelt werden, ob die Ausführung innerhalb der Visual Basic-Umgebung erfolgt.

Ihre Aufgabe besteht darin, den Grund dafür herauszufinden.

Puzzle 6

Wo bleibt das Icon?

Die meisten Programmierer kennen sich mit Icons aus – diesen kleinen niedlichen rechteckigen Icons, die in Windows verwendet werden.[7] Icons können in speziellen Icondateien (üblicherweise mit der Erweiterung **.ico**), als Ressourcen innerhalb von Windows oder in DLLs gespeichert werden.

Windows stellt verschiedene integrierte Icons bereit, die mit Hilfe der Funktion `LoadIcon` geladen werden können. In der Win32-Dokumentation wird diese Funktion folgendermaßen definiert:

```
HICON LoadIcon(
   HINSTANCE hInstance, // Zugriffsnummer für Anwendungsinstanz
   LPCTSTR lpIconName   // Iconnamenzeichenfolge oder Ressourcenbezeichner
   );
```

Diese Beschreibung erscheint sehr einfach. Wie jede Zugriffsnummer wird der `hInstance`-Parameter als Typ `Long` definiert und als Wert übergeben. Bei dem Parameter `IpIconName` handelt es sich um eine Zeichenfolge. Der Typ lässt sich folgendermaßen beschreiben:

LP (longfar): Zeiger.

C Konstant (Constant): Dies bedeutet, dass die API-Funktion den Inhalt der Zeichenfolge nicht verändert.

T Variiert in Abhängigkeit vom Einsprungpunkt. Als Format für die Zeichenfolge wird für den ANSI-Einsprungpunkt ANSI, für den Unicode-Einsprungpunkt Unicode verwendet.

STR Auf NULL endende C-Zeichenfolge.

Die Deklaration lautet also:

```
Private Declare Function LoadIcon Lib "user32.dll" Alias "LoadIconA" _
   ( ByVal hInstance As Long, _
   ByVal lpIconName As String) As Long
```

7. Wieso rechteckig? Icons sind immer rechteckig – es ist nur so, dass Teile eines Icons transparent angezeigt werden können, so dass der Eindruck entsteht, Icons könnten viele verschiedene Formen aufweisen.

Wenn Sie eine Iconressource laden, enthält der Parameter `hInstance` die Modulzugriffsnummer der geladenen DLL. Ist der `hInstance`-Parameter NULL, wird ein SystemIcon geladen. Der `IpIconName`-Parameter gibt das zu ladende Icon an und verfügt außerdem über die folgende interessante Beschreibung:

Er zeigt auf eine in NULL endende Zeichenfolge, die den Namen der zu ladenden Iconressource enthält. Alternativ kann dieser Parameter den Ressourcenbezeichner in **low-order word** und 0 in **high-order word** enthalten. Verwenden Sie zum Erstellen dieses Wertes das Makro `MAKEINTRESOURCE`.

Die Funktion `MAKEINTRESOURCE` wird in der Headerdatei `winuser` folgendermaßen definiert:

```
#define MAKEINTRESOURCEA(i) (LPSTR)((DWORD)((WORD)(i)))
#define MAKEINTRESOURCEW(i) (LPWSTR)((DWORD)((WORD)(i)))
#ifdef UNICODE
#define MAKEINTRESOURCE  MAKEINTRESOURCEW
#else
#define MAKEINTRESOURCE  MAKEINTRESOURCEA
#endif // !UNICODE
```

Wenn der Parameterwert für `hInstance` NULL lautet, kann es sich bei dem `IpIconName`-Parameter um eine der folgenden vordefinierten Konstanten handeln:

```
#ifdef RC_INVOKED
#define IDI_APPLICATION     32512
#define IDI_HAND            32513
#define IDI_QUESTION        32514
#define IDI_EXCLAMATION     32515
#define IDI_ASTERISK        32516
#if(WINVER >= 0x0400)
#define IDI_WINLOGO         32517
#endif /* WINVER >= 0x0400 */
#else
#define IDI_APPLICATION     MAKEINTRESOURCE(32512)
#define IDI_HAND            MAKEINTRESOURCE(32513)
#define IDI_QUESTION        MAKEINTRESOURCE(32514)
#define IDI_EXCLAMATION     MAKEINTRESOURCE(32515)
#define IDI_ASTERISK        MAKEINTRESOURCE(32516)
#if(WINVER >= 0x0400)
#define IDI_WINLOGO         MAKEINTRESOURCE(32517)
#endif /* WINVER >= 0x0400 */
#endif /* RC_INVOKED */
```

Abbildung P6-1 Das Hauptformular der Anwendung **IconView**

Abbildung P6-1 zeigt das Hauptformular der Anwendung **IconView** so, wie es normalerweise angezeigt werden sollte. Über jede Optionsschaltfläche wird ein anderes vordefiniertes Icon ausgewählt.

Die Variable `m_IconID` im **IconView**-Projekt (auf der Begleit-CD-ROM) enthält den dem jeweiligen Icon entsprechenden Konstantenwert. Während des Ereignisses `picture box Paint` wird die Funktion `LoadIcon` dazu verwendet, die aktuelle Iconressource zu laden. Die `DrawIcon`-Funktion wird zum Zeichnen der Icons im Bildfeld verwendet. Die `DrawIcon`-Funktion wird in der Win32-Dokumentation folgendermaßen definiert:

```
BOOL DrawIcon(
   HDC hdc,                  // Zugriffsnummer für Gerätekontext
   int X,                    // x-Koordinate des linken oberen Bildpunktes
   int Y,                    // y-Koordinate des linken oberen Bildpunktes
   HICON hIcon               // Zugriffsnummer des zu zeichnenden Icons
   );
```

Der Parameter `hDC` stellt den Gerätekontext des Fensters dar, in dem Sie zeichnen werden, über die Parameter `X` und `Y` wird der Iconstandort angegeben, und durch den `hIcon`-Parameter wird die Zugriffsnummer für das Icon angegeben – sämtliche 32 Bits, alle als Wert.

Hier das Beispielprogramm:

```
' IconView Puzzle
' Copyright © 1998 by Desaware Inc. Alle Rechte vorbehalten

Option Explicit

Private Declare Function LoadIcon Lib "user32.dll" Alias _
```

```
   "LoadIconA" ( _
   ByVal hInstance As Long, _
   ByVal lpIconName As String) As Long

Private Declare Function DrawIcon Lib "user32" (ByVal hdc As Long, _
ByVal x As Long, ByVal y As Long, ByVal hIcon As Long) As Long

Private Const IDI_APPLICATION = 32512&
Private Const IDI_HAND = 32513&
Private Const IDI_QUESTION = 32514&
Private Const IDI_EXCLAMATION = 32515&
Private Const IDI_ASTERISK = 32516&
Private Const IDI_WINLOGO = 32517&

Private m_IconID As Long

Private Sub Form_Load()
   m_IconID = IDI_APPLICATION
End Sub

Private Sub optIcon_Click(Index As Integer)
   Select Case Index
Case 0
       m_IconID = IDI_APPLICATION
     Case 1
       m_IconID = IDI_ASTERISK
     Case 2
       m_IconID = IDI_EXCLAMATION
     Case 3
       m_IconID = IDI_HAND
     Case 4
       m_IconID = IDI_QUESTION
     Case 5
       m_IconID = IDI_WINLOGO   End Select
   Picture1.Refresh
End Sub


Private Sub Picture1_Paint()
   Dim iconhandle As Long
   iconhandle = LoadIcon(0, m_IconID)
```

```
    If iconhandle = 0 Then
      MsgBox GetErrorString(Err.LastDllError)
    Else
      Call DrawIcon(Picture1.hdc, 0, 0, iconhandle)
    End If
End Sub
```

Ergebnisse

Es werden keine Icons angezeigt. Woran liegt das?

Puzzle 7

Überladene Grafiken

Zu den nützlichsten Win32-API-Funktionen gehören verschiedene Grafikfunktionen, die die Funktionen von Visual Basic mehr als in den Schatten stellen. Viele dieser Win32-Grafikfunktionen stellen nicht nur komplexe Zeichen- und Füllfunktionen bereit, sondern verfügen außerdem über eine enorme Ausführungsgeschwindigkeit. Dies ist ein Grund, weshalb in Windows viele Zeichenoperationen an die Grafikkarte oder Druckerengine weitergeleitet werden können, anstatt die Grafiken auf Ihrem Computer zu verarbeiten.

In diesem Beispiel wenden wir uns einer der einfacheren Grafikfunktionen zu, nämlich der Polyline-Funktion. In der Win32 SDK wird diese Funktion folgendermaßen definiert:

```
BOOL Polyline(
   HDC hdc,                    // Zugriffsnummer für Gerätekontext
   CONST POINT *lppt,          // Adresse des Arrays mit den Endpunkten
   int cPoints                 // Anzahl der Punkte im Array
   );
```

Daraus ergibt sich Folgendes:

- Der `hdc`-Parameter ist die Zugriffsnummer eines Gerätekontextes für Zeichenoperationen. Wie bei die meisten Zugriffsnummern handelt es sich um einen Wert vom Typ `Long`.
- Der `Ippt`-Parameter ist ein Array mit `POINT`-STRUKTUREN. Das `CONST`-Schlüsselwort gibt an, dass die Werte in diesem Array nicht durch die API-Funktion verändert werden können.
- Der `nCount`-Parameter stellt die Anzahl der Arrayeinträge dar, also die Anzahl der Linienpunkte. Durch die Funktion werden sämtliche Punkte des Arrays durch Linien in der aktuellen Farbe miteinander verbunden.

Eine `POINT`-Struktur wird in C folgendermaßen definiert:

```
typedef struct  tagPOINT
   {
   LONG x;
   LONG y;
   }  POINT;
```

Die POINT-Struktur sollte in Visual Basic wie folgt deklariert werden:

```
Type POINTAPI
     x As Long
     y As Long
End Type
```

Warum wird der Name von POINT in POINTAPI geändert? Da es sich bei POINT um ein Wort handelt, das für Visual Basic reserviert ist, kann es bei einer Ersetzung dieses Wortes durch eine benutzerdefinierte Struktur zu Missverständnissen kommen, nicht nur bei den Personen, die Ihr Programm verstehen möchten, sondern auch auf der Seite von Visual Basic selbst.

Das Polyline-Programm ist so konzipiert, dass ein Kreis in eine bestimmte Anzahl von Punkten unterteilt wird, die nach dem Zufallsprinzip durch Linien miteinander verbunden werden, wie in Abbildung P7-1 dargestellt wird.

Das Programm verfügt über ein Textfeld, in das Sie die Anzahl der Punkte eingeben können, die Schaltfläche **Löschen**, mit der Sie das Formular löschen können, die Schaltfläche **Draw Shape**, mit der Sie die Zeichenoperation ausführen, und das Kontrollkästchen **Auto**, über welches das Programm nach dem Zufallsprinzip die Punkteanzahl auswählt und die Linien basierend auf diesen Punkten einzeichnet. Auf diese Weise werden Spezialeffekte erzielt, die eine fast hypnotische Wirkung haben.

Das nachstehende Beispiel zeigt das Polyline-Programm. Das Nachvollziehen der Funktion SetupPoints kann sich als etwas schwierig erweisen, wenn Sie sich nicht einige trigonometrische Berechnungen aus Ihrer Schulzeit ins Gedächtnis zurückrufen. Es ist zur Lösung dieses Puzzles zwar nicht unbedingt erforderlich, die mathematischen Berechnungen zu verstehen, die sich hinter der Zeichenoperation verbergen, nachfolgend werde ich jedoch all denen eine kurze Erläuterung geben, die diesen Versuch wagen möchten.

Abbildung P7-2 wird Ihnen dabei helfen, die folgende Logik zu verstehen.

Im nachstehenden Beispiel weist die Form eine größere Breite als Höhe auf. Das bedeutet, dass der Radius des Kreises der halben Höhe entspricht, also ScaleHeight/2. Der vertikale Abstand zwischen dem Kreismittelpunkt und einem beliebigen Punkt auf der Kreislinie entspricht dem Radius multipliziert mit sin(A). Der Kreismittelpunkt ist (ScaleWidth/2, ScaleHeight/2), d.h. diese Werte müssen zum Abstand vom Kreismittelpunkt addiert werden, um die Position des Punktes im Formular zu berechnen. Der ScaleMode-Parameter für das Formular wird auf Pixel eingestellt, da die API-Funktion Polyline die Angabe der Koordinatenwerte in Pixeln erwartet.

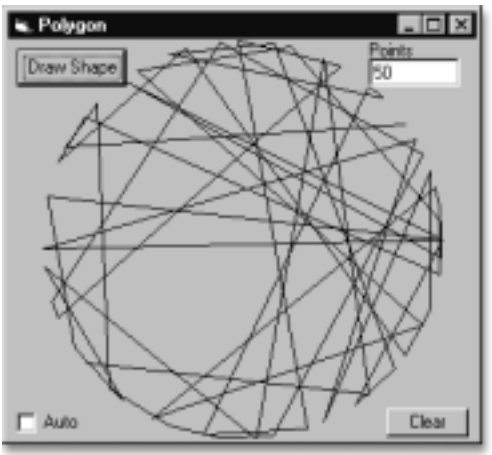

Abbildung P7-1 Das Polyline-Programm verbindet die Punkte eines Kreises nach dem Zufallsprinzip durch Linien miteinander

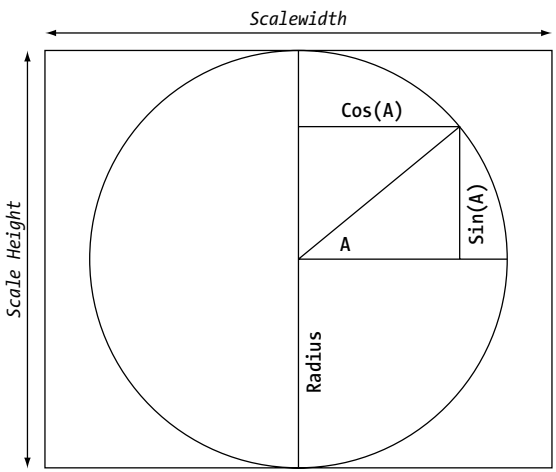

Abbildung P7-2 Darstellung der Kreisberechnung

```
' Polyline Example
' Copyright © 1998 by Desaware Inc. Alle Rechte vorbehalten

Option Explicit

Private Type POINTAPI
     X As Long
     y As Long
End Type
```

```
Private Declare Function Polyline Lib "gdi32" (ByVal hdc As Long, _
lppt() As POINTAPI, ByVal nCount As Long) As Long

Private PointArray() As POINTAPI

Private CurrentColorIndex As Integer

' Wir benötigen die Arccos-Funktion - arccos(0) ist Pi
Private Function Arccos(X As Double)
   Arccos = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1)
End Function
' Diese Funktion unterteilt den Kreis in Punkte und verteilt diese beliebig
Private Sub SetupPoints(PointCount As Long, Radius As Long, Xoffset As _
Long, Yoffset As Long)
   Dim AngleIncrement As Double
   Dim PointNumber As Long
   Dim SwapPoint As Long
   Dim TempPoint As POINTAPI

   ' Arccos(0) ist Pi. 2 * Pi stellt die Anzahl der Radianten im Kreis dar.
   ' 2 * Pi /
 PointCount stellt den Winkel (in Radianten) zwischen den Punkten dar.
   AngleIncrement = 2 * Arccos(0) / PointCount
   ReDim PointArray(PointCount)
   ' Die Cos- und Sin-Funktionen erhalten die x- und y-
Werte für die Punktpositionen entlang des Kreises
   ' bei einem vorgegebenen Winkel.
   For PointNumber = 0 To PointCount - 1
     PointArray(PointNumber).X = Cos(PointNumber * PointCount) * _
     Radius + Xoffset
     PointArray(PointNumber).y = Sin(PointNumber * PointCount) * _
     Radius + Yoffset
   Next PointNumber
   ' Zurückkehren an den ursprünglichen Punkt
   LSet PointArray(PointCount) = PointArray(0)

   ' Ersten oder letzten Punkt nicht verschieben
   For PointNumber = 1 To PointCount
     ' Jeden Punkt durch einen zufällig ermittelten Punkt ersetzen
     SwapPoint = Int(Rnd() * PointCount)
     LSet TempPoint = PointArray(SwapPoint)
```

```
      LSet PointArray(SwapPoint) = PointArray(PointNumber)
      LSet PointArray(PointNumber) = TempPoint
   Next PointNumber
End Sub


Private Sub chkAuto_Click()
   Timer1.Enabled = chkAuto.Value
End Sub

Private Sub cmdClear_Click()
   ' Form löschen
   Me.Cls
End Sub

Private Sub cmdDraw_Click()
   Dim points As Long
   Dim Radius As Long

   ' Abrufen der Punkteanzahl vom Textfeld
   points = txtPoints.Text
   If points = 0 Then points = 2
   ' Der Radius ist kleiner als die halbe Höhe oder Breite
   If ScaleWidth < ScaleHeight Then
     Radius = ScaleWidth / 2
   Else
     Radius = ScaleHeight / 2
   End If

   SetupPoints points, Radius, ScaleWidth / 2, ScaleHeight / 2
   Call Polyline(hdc, PointArray(), points)
   ' Zur nächsten der 16 Standardfarben wechseln
   CurrentColorIndex = (CurrentColorIndex + 1) Mod 16
   Me.ForeColor = QBColor(CurrentColorIndex)
End Sub

Private Sub Form_Load()
   Randomize
End Sub

Private Sub Timer1_Timer()
```

```
    ' Zeichenoperation mit den aktuellen Einstellungen durchführen
    cmdDraw_Click
    ' Neuen Zählerstand für nächsten Durchlauf einstellen
    txtPoints.Text = Int(Rnd() * 50) + 5
End Sub
```

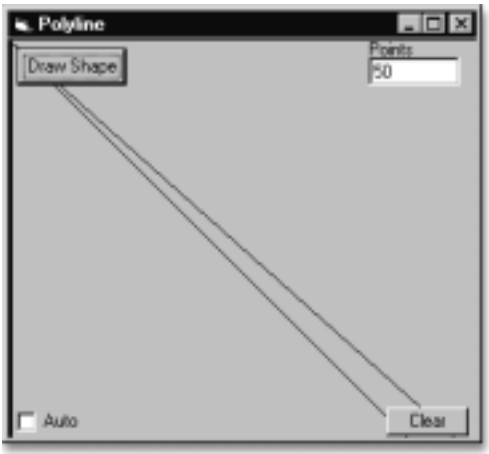

Abbildung P7-3 Das aktuell mit dem Programm gezeichnete Bild entspricht nicht dem gewünschten Ergebnis

Ergebnisse

Die `Polyline`-Funktion arbeitet sehr schnell. Sie erkennen dies, wenn Sie auf die Schaltfläche **Auto** klicken und beobachten, wie die Zeichnung automatisch aktualisiert wird.

Dies sollte eigentlich passieren, wenn Sie auf die Schaltfläche **Draw Shape** klicken. Die tatsächlichen Ergebnisse variieren allerdings von einem leeren Formular bis zu einer Zeichnung, die der in Abbildung P7-3 gezeigten ähnelt.

Können Sie das Programm reparieren?

Puzzle 8

Bockspringen

Windows enthält die API-Funktion `Getversionex`, mit der Sie über Ihre Anwendung die Version des derzeit laufenden Betriebssystems ermitteln können. Die Information über die Betriebssystemversion mag unwichtig erscheinen und ist auch unerheblich für jene Entwickler, die ausschließlich Visual Basic verwenden. Die Systemversionsangabe kann jedoch für Programmierer, die die Win32-API verwenden, äußerst wichtig sein. Warum? Weil Microsoft dem Betriebssystem ständig neue API-Funktionen hinzufügt. Über die Versionsinformation können Sie vor dem Aufruf einer Funktion sicherstellen, dass diese auch tatsächlich verfügbar ist.

Die Leistung einer Win32-Anwendung wird unter einem älteren Betriebssystem üblicherweise erheblich eingeschränkt. Mit anderen Worten: Wenn Sie die Funktionalität neuerer Systeme wie z.B. Windows 2000 voll nutzen möchten, sollten Sie über Ihre Anwendung keine Funktionen aufrufen, die von einem älteren System nicht unterstützt werden. Statt dessen sollten Sie diesen Teil der Anwendung deaktivieren, einen alternativen Ansatz zur Bereitstellung der jeweiligen Funktionalität verwenden oder dem Benutzer den Vorschlag machen, sein System aufzurüsten.

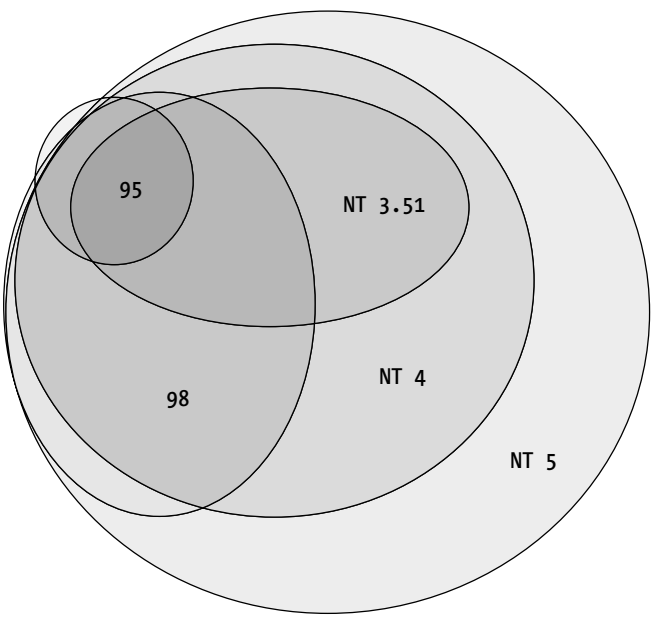

Abbildung P8-1 Jede Betriebssystemversion unterstützt unterschiedliche Win32-API-Funktionen

Unter den derzeit verwendeten Betriebssystemen (nein, wir werden die 16-Bit-Systeme nicht länger berücksichtigen, auch wenn diese z. T. noch verwendet werden) stellen Windows 95 und NT 3.51 die Systeme mit der niedrigsten Funktionalität dar. Bei beiden Systemen können verschiedene unter NT 4, und Windows 98 verfügbare Funktionen nicht verwendet werden bzw. werden nicht unterstützt. In Abbildung P8-1 sind die sich überschneidenen Funktionen zu sehen. Wie Sie erkennen können, baut jede Windows NT-Version auf dem API-Satz einer früheren Version auf. Windows NT, Version 4.0, verfügt über die meisten der Funktionen von Windows 95 und Windows 98, umgekehrt lässt sich dies leider nicht behaupten. Windows 98 unterstützt mehr NT-Funktionen als Windows 95, bei weitem jedoch nicht alle.

In Win32-Anwendungen wird die Funktion `GetVersionEx` dazu verwendet, detaillierte Informationen über das Betriebssystem abzurufen.

In der Windows-Dokumentation wird diese Funktion folgendermaßen definiert:

Mit der `GetVersionEx`-Funktion können erweiterte Informationen zur derzeit ausgeführten Betriebssystemversion abgerufen werden.

```
BOOL GetVersionEx(
  LPOSVERSIONINFO lpVersionInformation   // Zeiger auf Version
                                        // Informationsstruktur
);
```

Die Funktion verwendet als Parameter einen Zeiger auf eine `OSVERSIONINFO`-Struktur, die folgendermaßen definiert wird:

```
typedef struct _OSVERSIONINFO{
DWORD dwOSVersionInfoSize;          ' Größe der Struktur
DWORD dwMajorVersion;               ' Hauptversionsnummer
DWORD dwMinorVersion;               ' Unterversionsnummer
DWORD dwBuildNumber;                ' Build-Nummer
DWORD dwPlatformId;                 ' Plattformbezeichner
TCHAR szCSDVersion[ 128 ];          ' Zeichenfolge mit zusätzlichen
                                    ' Informationen
}  OSVERSIONINFO;
```

Bei der Plattform-ID handelt es sich um eine der folgenden Konstanten:

`VER_PLATFORM_WIN32_WINDOWS`	Win32 unter Windows 95 oder Windows 98
`VER_PLATFORM_WIN32_NT`	Win32 unter Windows NT
`VER_PLATFORM_WIN32_CE`	Win32 unter Windows CE

Der Unterversionsnummer lautet bei Windows 95 0, bei Windows 98 lautet sie 10. Die Hauptversionsnummer lautet sowohl für Windows 95 als auch für Windows 98 4.

Sie können zunächst versuchsweise den folgenden Code verwenden:

```
' WinVer Project
' Copyright © 1998 by Desaware Inc. Alle Rechte vorbehalten

Private Type OSVERSIONINFO
   dwOSVersionInfoSize As Long
   dwMajorVersion As Long
   dwMinorVersion As Long
   dwBuildNumber As Long
   dwPlatformId As Long
   szCSDVersion As String * 128
End Type

Private Declare Function GetVersionEx Lib "kernel32" _
(os As OSVERSIONINFO) As Long

Private Const VER_PLATFORM_WIN32s = 0
Private Const VER_PLATFORM_WIN32_WINDOWS = 1
Private Const VER_PLATFORM_WIN32_NT = 2


Option Explicit

Private Sub Form_Load()
   Dim os As OSVERSIONINFO
   Dim res As Long
   Dim s$
   Dim nullpos&
   res = GetVersionEx(os)
   If res <> 0 Then ' success
     Select Case os.dwPlatformId
       Case VER_PLATFORM_WIN32_NT
         s$ = "Windows NT version " & os.dwMajorVersion _
         & os.dwMinorVersion
       Case VER_PLATFORM_WIN32_WINDOWS
         s$ = "Windows "
         If os.dwMinorVersion = 10 Then s$ = s$ & "98" Else _
```

```
            s$ = s$ & "95"
          Case Else
            s$ = "Unknown OS"
        End Select
        s$ = s$ & vbCrLf
        s$ = s$ & "Build: " & os.dwBuildNumber & vbCrLf
        nullpos = InStr(os.szCSDVersion, Chr$(0))
        If nullpos > 1 Then s$ = s$ & Left$(os.szCSDVersion, nullpos - 1)
      Else
        s$ = "GetVersionInfoEx failed"
      End If
      Label1.Caption = s$
End Sub
```

Ergebnisse

Laufzeitfehler 453: DLL-Einsprungpunkt `GetVersionEx` in `kernel32` nicht gefunden.

Können Sie das Programm reparieren?

Teil II
Die Lösungen

In diesem Teil des Buches finden Sie die Lösungen zu den Puzzlen aus Teil I. Bevor Sie weiterlesen, sollten Sie bedenken, dass der erzielte Lerneffekt direkt proportional zu der Zeit ist, die Sie darauf verwenden, die Puzzle selbst zu lösen.

Wenn Sie also bisher noch keines der Puzzle gelöst haben, schlage ich vor, die folgenden Schritte auszuführen, bevor Sie sich die Lösungen anschauen.

- Lesen Sie die Hinweise in Anhang A.
- Arbeiten Sie die Tutorien in Teil III durch, da Sie mit deren Hilfe die Fähigkeit erlangen können, die Puzzle selbständig zu lösen.

Dies kann ein wenig zeitaufwendiger sein, ist aber umso effektiver.

Die Lösungen finden Sie – natürlich im Buch.

Das »total gemeine« Verlagsteam von

Galileo Press

Lösung 1

Wo steckt denn jetzt der API-Aufruf?

Die Deklaration Erschien vom Aufbau her vielleicht sinnvoll, sie enthielt jedoch zwei Fehler. Obwohl die Funktion als `GetUserDefaultLCID` bezeichnet werden kann, stehen Ländereinstellungen doch in Zusammenhang mit dem Betriebssystem und sind daher Teil der `kernel32`-DLL und nicht, wie in der ursprünglichen Deklaration dargestellt, der `User32`-DLL.

Darüber hinaus muss bei den Win32-Funktionen die Groß- und Kleinschreibung berücksichtigt werden (im Gegensatz zu den 16-Bit-Windows-Funktionen). Bei der Mehrzahl der API-Funktionsnamen, die mehrere Wörter enthalten, wird der erste Buchstabe eines jeweiligen Wortes groß geschrieben, wie dies bei der `GetWindowText`-Funktion der Fall ist. Bei `LCID` handelt es sich jedoch um ein Akronym (`LoCale IDentifier`), daher müssen alle Buchstaben groß geschrieben werden.

Diese Art von Fehlern werden leicht übersehen. Das Gute hierbei ist, dass diese Fehler in Visual Basic nicht nur ermittelt werden können, sondern Sie sogar eine eindeutige Meldung diesbezüglich erhalten. Wenn Sie den Laufzeitfehler 453 empfangen (Angegebene DLL-Funktion nicht gefunden), wird das Problem mit großer Sicherheit durch die Deklaration verursacht, häufig durch einen Fehler im Funktions- oder Bibliotheksnamen.

Durch den nachstehenden Code wird nicht nur das Problem behoben, sondern gleichzeitig die Verwendung der `LCID` zur Ermittlung des englischen Namens der durch diese Ländereinstellung verwendeten Sprache verdeutlicht.

```
' Wo steckt denn jetzt der API-Aufruf?
' Copyright © 1998 by Desaware Inc. Alle Rechte vorbehalten

Option Explicit

Private Declare Function GetUserDefaultLCID Lib "kernel32" () As Long

Private Const LOCALE_SENGLANGUAGE = &H1001&
'  Englischer Name der Sprache
Private Declare Function GetLocaleInfo Lib "kernel32" Alias _
"GetLocaleInfoA" (ByVal Locale As Long, ByVal LCType As Long, _
ByVal lpLCData As String, ByVal cchData As Long) As Long

Private Sub Command1_Click()
   Dim lcid&
```

```
    Dim info$
    lcid = GetUserDefaultLCID()
    info = String$(255, 0)
    Call GetLocaleInfo(lcid, LOCALE_SENGLANGUAGE, info, 255)
    info$ = Left$(info, InStr(info, Chr$(0)) - 1)
    MsgBox lcid & ":" & info$, vbOKOnly, "User Default LCID"
End Sub
```

Index

C

D

E

F

G

H

I

K

P

R

S

T

U

V

W