Kapitel 25

Versionsverwaltung

Windows 3.0. Windows 3.1. Windows 95. Windows 98. Windows NT 3.51. Windows NT 4. Visual Basic 1. Visual Basic 2. Visual Basic 3. Visual Basic 4. Visual Basic 5. Visual Basic 6.

Wir sind daran gewöhnt, daß wir es ständig mit verschiedenen Versionen von Softwarepaketen zu tun haben. Als Kunden erwarten wir neue Funktionen, wenn ein Paket eine neue Versionsnummer hat. So sahen die Entwickler ihre Softwarepakete, aber das gilt jetzt nicht mehr.

Eine typische Windows-Applikation ist von vielen verschiedenen Komponenten abhängig. Sie verwendet Funktionen, die von dynamischen Link-Bibliotheken bereitgestellt werden, die Teil des Betriebssystems sind. Sie verwendet Funktionen, die von dynamischen Link-Bibliotheken bereitgestellt werden, die Teil der OLE-Implementierung des Betriebssystems sind. Sie verwendet Standardfensterklassen, die von weiteren dynamischen Link-Bibliotheken bereitgestellt werden, beispielsweise der Bibliothek für die Standardsteuerelemente und die Standarddialoge. Sie verwendet Systemdienste wie etwa Mail und Winsock, die von anderen dynamischen Link-Bibliotheken implementiert werden. Sie kann ActiveX-Steuerelemente verwenden, die alle durch eine dynamische Link-Bibliothek implementiert werden (in der Regel mit der Dateinamenerweiterung `.OCX`). Für eine Visual-Basic-Applikation braucht man immer die Laufzeit-DLL von Visual Basic und unter Umständen zusätzliche Laufzeit-DLLs, die von den verschiedenen ActiveX-Steuerelementen verwendet werden. Und jetzt ist Ihr Programm auch von neuen ActiveX-Komponenten oder Steuerelementen abhängig, die Sie für die Verwendung in der Applikation entwickeln.

Jede Menge Abhängigkeiten. Und wirklich jede dieser Komponenten hat eine eigene Version, die völlig unabhängig von der Versionsnummer der eigentlichen Applikation ist. Wenn Sie Ihre Applikation entwickeln, lassen Sie sie auf einer bestimmten Komponentenmenge basieren und testen sie damit. Wenn Sie versuchen, Ihre Applikation auf einem System auszuführen, wo auch nur eine dieser Komponenten veraltet ist, verhält sich Ihr Programm möglicherweise nicht mehr wie gewünscht. Das ist ein grundlegendes Problem bei komponentenbasierter Software, das wirklich ernst zu nehmen ist.

In diesem Kapitel erfahren Sie, wie man damit zurechtkommt, sowohl in Hinblick auf die Entwicklung von Komponenten als auch ihre Veröffentlichung.

25.1 Versionsressourcen

Ganz offensichtlich ist es für die Lösung dieses Problems erforderlich, jede Komponente mit einer Versionsnummer zu markieren. Windows erlaubt, daß in jede Programmdatei oder dynamischen Link-Bibliothek eine Versionsressource eingebettet wird. Diese Ressource enthält die Versionsnummer und eine beschreibende Information.

Abbildung 25.1 zeigt das Dialogfeld für die Einstellung der Projekteigenschaften. Sie können beschreibenden Text angeben, unter anderem den Firmennamen, die Dateibeschreibung, den Produktnamen sowie Copyright- und Markeninformation. Sie sollten jedoch immer eine Versionsnummer setzen.

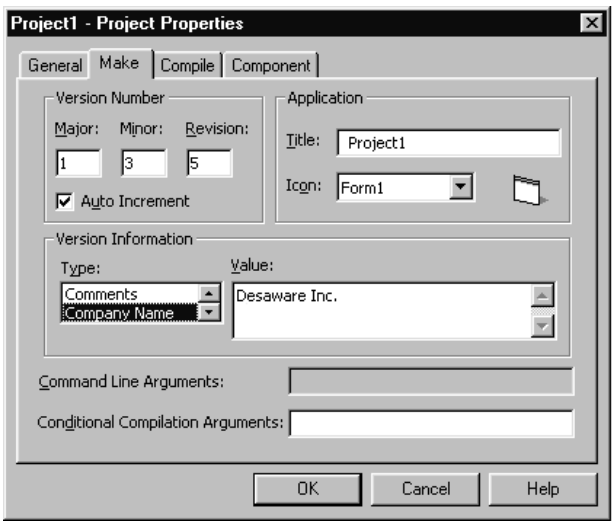

Abb. 25.1: Das Dialogfeld Projekteigenschaften

Als erstes sollten Sie immer das Kontrollkästchen AUTOMATISCH ERHÖHEN markieren. Damit ist sichergestellt, daß bei jeder neuen Kompilierung die Versionsnummer Ihrer Komponente aktualisiert wird.

Warum ist das so wichtig? Weil die Versionsressource fast von jedem Installationsprogramm verwendet wird, um zu verhindern, daß neuere Komponenten durch ältere Komponenten überschrieben werden. Sie sehen, Versionsressourcen markieren eine Komponente oder eine Programmdatei, aber es gibt nichts im System oder in Visual Basic, das automatisch prüft, ob die abhängigen Komponenten vorhanden und korrekt sind, wenn eine Applikation ausgeführt wird. Und noch schlimmer ist, daß es häufig nicht einmal möglich ist, festzustellen, ob ein Problem durch eine Komponente verursacht wurde. Beispielsweise könnte ein Komponentenfehler, der verhindert, daß ein Steuerelement oder ein ActiveX-Dokument geladen wird, einfach einen Laufzeitfehler verursachen, der dem Container dann als HTML- oder Skriptingfehler berichtet wird.

Eine Komponente kann auf mehrere Arten von einer älteren Komponente überschrieben werden. Einige Installationsprogramme weisen darauf hin, wenn die Gefahr besteht, daß eine neuere Komponente überschrieben wird, und überlassen dem Benutzer die Entscheidung. Einige Installationsprogramme prüfen die Versionsnummern nicht korrekt. Manchmal werden Dateien einfach nur von einem System auf ein anderes kopiert.

Beachten Sie, daß die verwendeten Komponenten bei der OLE-Technologie von dem Inhalt der System-Registrierung abhängig sind. Jede Applikation kann eine ältere Version einer Komponente registrieren und damit letztlich eine neuere Version ersetzen, ohne sie zu überschreiben. Die alte Lösung, Komponenten in einem privaten Verzeichnis aufzubewahren, funktioniert nicht mehr.

All diese Szenarien zeigen, daß in einem System jederzeit veraltete Komponenten auftauchen können; und sie können die korrekte Arbeitsweise Ihrer Applikation stören, auch wenn diese korrekt installiert wurde. Es gibt mehrere Ansätze, mit diesem Problem zurechtzukommen:

- Installieren Sie Ihre Applikation neu, sobald sie ein ungewöhnliches Verhalten aufweist oder nicht mehr fehlerlos ausgeführt wird.
- Erzwingen Sie, daß Ihre Komponente oder Applikation eine eigene Überprüfung der Abhängigkeiten vornimmt. Das Win32-API beinhaltet mehrere Funktionen, die Ihnen erlauben, Versionsinformation aus beliebigen Programmdateien oder Komponenten zu lesen, die eine Versionsressource besitzen.
- Verwenden Sie einen Toolkit zur Überprüfung von Komponenten, wie beispielsweise den VersionStamper von Desaware. Er kann genutzt werden, um die Abhängigkeiten für Ihre Komponente oder Applikation zu überprüfen, und beinhaltet die Möglichkeit, Komponenten über das Internet oder ein Firmen-Intranet zu aktualisieren.
- Ignorieren Sie das Problem und machen Sie sich auf zeitaufwendige Support-Anrufe gefaßt, sobald es auftritt.

Sie sollten entscheiden, welche dieser Strategien Sie verfolgen möchten, um Ihre eigenen komponentenbasierten Applikationen zu veröffentlichen. Das Problem ist nicht so tragisch für Steuerelemente und Komponenten, die von Web-Seiten heruntergeladen werden. Der Browser übernimmt nämlich die Rolle eines Installationsprogramms und lädt die neuesten Komponenten herunter, sobald sie benötigt werden. Wie Sie jedoch bereits gesehen haben, ist das keine zuverlässige Lösung, weil der Browser keine aussagekräftigen Informationen bereitstellt, wenn während des Herunterladens irgend etwas schief läuft. Unabhängig davon, welchen Ansatz Sie für die Veröffentlichung Ihrer Komponenten wählen, ist klar, daß Sie als Komponentenentwickler eine Verantwortung gegenüber Ihren Benutzern tragen, und sicherstellen müssen, daß Ihre Komponenten korrekt markiert sind.

25.2 Versionskompatibilität

Betrachten wir das folgende Szenario. Sie haben eine Applikation entwickelt, die Version 1.0 des Steuerelements `Xyz.OCX` verwendet. Der Hersteller, der das Steuerelement `Xyz.OCX` entwickelt hat, gibt die neue Version 2.0 heraus. Er hat dabei jedoch eine der Eigenschaften des Steuerelements weggelassen, weil er dachte, sie sei nicht besonders praktisch.

Ihre Applikation läuft weiterhin korrekt, weil Sie immer noch Version 1.0 des Steuerelements verwenden. Irgendwann installieren Sie jedoch eine weitere Applikation, welche die Version 2.0 des Steuerelements verwendet. Das Installationsprogramm für diese Applikation erkennt Version 1.0 auf Ihrem System und ersetzt sie durch die aktualisierte Version. Wenn Sie Ihre alte Applikation nun zum nächsten Mal ausführen, verwendet sie Version 2.0 des Steuerelements. Sobald sie jedoch versucht, auf die fehlende Eigenschaft zuzugreifen, schlägt das Steuerelement fehl, möglicherweise mit einem Speicherfehler.

Die Versionsnummer in der Versionsressource ist nur eine Bezeichnung. Sie hat keine Funktionalität. Nichts garantiert, daß Version 2.0 einer Komponente erfolgreich in einer Applikation eingesetzt werden kann, die mit Version 1.0 einer Komponente kompiliert wurde. Das gesamte System der Komponenten-Upgrades setzt jedoch voraus, daß eine neuere Version einer Komponente alle Funktionen einer älteren Version unterstützt.

Als Komponentenentwickler müssen Sie sicherstellen, daß Ihre aktualisierten Komponenten abwärtskompatibel mit älteren Versionen sind. Das bedeutet, eine aktualisierte Komponente muß die folgenden Bedingungen erfüllen:

- Sie muß dieselben CLSID-Komponentenbezeichner und Typbibliothek-IDs wie die frühere Version haben.
- Sie muß alle zuvor unterstützten Eigenschaften, Methoden und Ereignisse unterstützen.

Darüber hinaus muß in der aktualisierten Komponente das folgende gelten:

- Alle Eigenschaften, Methoden und Ereignisse müssen dieselbe Dispatch-ID-Nummer (Prozedur-ID) haben.
- Alle Eigenschaften, Methoden und Ereignisse müssen dieselben Parameter und Parametertypen haben.
- Alle Eigenschaften, Methoden und Ereignisse müssen dieselbe Funktionalität aufweisen wie die frühere Version.

Wir werden gleich darauf zurückkommen, wie das zu bewerkstelligen ist.

Wenn Sie beschließen, keine Abwärtskompatibilität zu realisieren, sollten Sie wie folgt vorgehen:

- Ändern Sie den Projektnamen auf der Registerkarte ALLGEMEIN des Dialogfelds PROJEKTEIGENSCHAFTEN.
- Deaktivieren Sie die Kompatibilität, indem Sie auf der Registerkarte KOMPONENTE im Dialogfeld PROJEKTEIGENSCHAFTEN die Option KEINE KOMPATIBILITÄT setzen.
- Kompilieren Sie das Projekt neu, und ändern Sie den Namen der ausführbaren Datei oder DLL.

Wie Microsoft mit dieser Situation umgegangen ist, sehen Sie in der Geschichte von Visual Basic. Als man von Version 3 zu Version 4 wechselte, wurde die Laufzeitdatei, die von Visual Basic genutzt wurde, von VBRUN300.DLL in VB40032.DLL geändert. Als Microsoft jedoch einen kleinen Upgrade von Visual Basic 4 vornahm, Version 4.0A, behielt es die Abwärtskompatibilität mit der früheren VB40032.DLL bei und änderte nur die Versionsnummer in der Ressourcendatei.

Wie können Sie Abwärtskompatibilität bewahren? Es ist einfach, dieselben Eigenschaften, Elemente und Ereignisse und ihre Parameter beizubehalten. Aber Visual Basic verbirgt die Zuweisung von CLSID-Werten und Prozedur-IDs. Das bedeutet, es muß eine Möglichkeit geben, Visual Basic mitzuteilen, daß Sie die vorhergehenden Werte beibehalten möchten. Das erfolgt auf der Registerkarte Komponente im Dialogfeld PROJEKTEIGENSCHAFTEN. Sie bietet drei Kompatibilitätstypen:

- Keine Kompatibilität
- Projekt-Kompatibilität
- Binär-Kompatibilität

Was KEINE KOMPATIBILITÄT bedeutet, ist offensichtlich. Visual Basic definiert den Typbibliotheksbezeichner und alle Schnittstellen- und Prozedurbezeichner neu, wenn das Projekt neu kompiliert wird.

Die PROJEKT-KOMPATIBILITÄT realisiert nur eine eingeschränkte Kompatibilität. Sie verwendet den aktuellen Typbibliotheksbezeichner und übernimmt CLSID und IID für alle Klassen und Schnittstellen, die mit der Referenzdatei kompatibel sind. Das ist ganz praktisch, weil man keine neuen Referenzen auf die Komponente einrichten muß, aber Visual Basic prüft nicht, ob das Projekt wirklich abwärtskompatibel ist.

Die Einstellung BINÄR-KOMPATIBILITÄT ist eine Methode, Visual Basic mitzuteilen, daß man sicherstellen will, daß die neu kompilierte Komponente abwärtskompatibel zu einer früheren Version der Komponente ist. Visual Basic verwendet denselben Typbibliotheksbezeichner und unterstützt existierende Schnittstellen- und Prozedurbezeichner. Wenn Sie eine Änderung vornehmen, die die Abwärtskompatibilität durchbrechen könnte, warnt Visual Basic Sie. Nehmen Sie diese Warnungen ernst! Wenn Sie die Änderungen dennoch vornehmen, kommt das der Auswahl von »Keine Kompatibilität« gleich. Visual Basic kann die Kompatibilität der Schnittstellen- und Prozedurdefinitionen beibehalten, aber Ihre Aufgabe ist es, die funktionale Kompatibilität sicherzustellen.

Hier kommt eine kurze Geschichte, die etwa vor fünf Jahren passiert ist, und die vielleicht verdeutlicht, was ich meine. Das erste Produkt von Desaware war die Custom Control Factory, ein flexibles Schaltflächen-Steuerelement, das die animierte Schaltfläche für Microsoft zur Einbindung in Visual Basic bereitstellte. Dieses Steuerelement hatte die Eigenschaft Frame, die auf den aktuell darzustel-

lenden Animations-Frame gesetzt werden konnte. Wenn man versuchte, die Eigenschaft auf eine negative Zahl zu setzen, wurde diese Einstellung einfach ignoriert und es trat kein Fehler auf. In der Rückschau wissen wir, daß wir in diesem Fall einen Fehler aufwerfen hätten sollen. Beherzigen Sie jedoch, daß das Steuerelement in den ersten Tagen der VBX-Technologie geschrieben wurde. Wir waren noch an das Gesamtkonzept der benutzerdefinierten Steuerelemente und Visual Basic gewöhnt, die Konventionen, denen wir heute unterliegen, waren noch nicht klar definiert.

Während einem der Upgrades – ich glaube, es war von Version 1 zu Version 2 – forderte uns Microsoft auf, eine Fehlerüberprüfung für die Eigenschaft auszuführen und einen Fehler aufzuwerfen, wenn sie auf eine negative Zahl gesetzt wurde. Ich habe protestiert, mit dem Argument, daß diese Änderung für den ohnehin unwahrscheinlichen Fall, daß ein Programm wirklich auf eine negative Zahl setzen sollte, die Abwärtskompatibilität zunichte machen würde. Aber Microsoft bestand auf der Änderung. Damals war es üblich, daß in solchen Fällen ein Fehler aufgeworfen werden sollte. Jedes Programm, das die Frame-Eigenschaft auf eine negative Zahl setzte, hatte aber überdies ohnehin selbst einen Fehler, und wäre wirklich ein Problem aufgetreten, wäre es Sache des anderen Programmierers gewesen.

Durch die Lizenzierung zahlte Microsoft aber für die Arbeit und hatte das Recht, Dinge zu fordern, so daß ich die Änderung in dieser Version des Steuerelements vornahm. Ich behielt jedoch das existierende Verhalten im vollständigen Custom Control Factory-Produkt bei, nur für den Fall. Man weiß ja nie. Es stellte sich heraus, daß es ein Programm gab, das einen Fehler enthielt, und das die Eigenschaft manchmal auf eine negative Zahl setzte. Es war eine der ersten Versionen von Microsoft Encarta! Stellen Sie sich das vor!

Die Moral von der Geschichte ist offensichtlich: Visual Basics Binär-Kompatibilitätsmodus tut sein bestes, um sicherzustellen, daß Ihre Schnittstellen und die Information in der Registrierung die Abwärtskompatibilität unterstützen. Aber die Beibehaltung der funktionalen Kompatibilität ist genau so wichtig, und die bleibt völlig Ihnen überlassen.

25.2.1 Interne Binär-Kompatibilität

Sehen wir uns noch einmal genauer an, was passiert, wenn Sie die Binär-Kompatibilität auswählen.

Das Projekt `gtpComp.vbp` im Beispielverzeichnis für Kapitel 25 auf Ihrer CD-ROM wurde genutzt, um die folgenden Beispiele zu erzeugen. Es enthält die endgültige Version des Projekts. Sie können dieser Entwicklung Schritt für Schritt folgen. Die darauffolgende Erklärung sollte aber völlig ausreichend sein, ohne daß Sie den gesamten Prozeß selbst nachvollziehen müssen.

`gtpComp.vbp` ist ein ActiveX-DLL-Projekt, das eine einzige Klasse enthält, `gtpComp`. Das Klassenmodul ist zunächst wie folgt definiert:

```
Public Sub A()
   Debug.Print "A called"
End Sub
```

Bei der Kompilierung wird die folgende Typbibliothek erzeugt:

```
// Generated .IDL file (by the OLE/COM Object Viewer)
//
// typelib filename: gtpComp.dll

[
  uuid(C7C07A2F-23DF-11D2-A754-00001C1AD1F8),
  version(1.0)
]
library gtpCompatibility
{
    // TLib :     // TLib : OLE Automation : {00020430-0000-0000-C000-000000000046}
    importlib("STDOLE2.TLB");

    // Forward declare all types defined in this typelib
    interface _gtpComp;

    [
      odl,
      uuid(C7C07A30-23DF-11D2-A754-00001C1AD1F8),
      version(1.0),
      hidden,
      dual,
      nonextensible,
      oleautomation
    ]
    interface _gtpComp : IDispatch {
        [id(0x60030000)]
        HRESULT A([in, out] short* Param);
    };

    [
      uuid(C7C07A31-23DF-11D2-A754-00001C1AD1F8),
      version(1.0)
    ]
    coclass gtpComp {
        [default] interface _gtpComp;
    };
};
```

Beachten Sie, daß die in diesem Listing aufgezeigten GUID-Werte möglicherweise nicht mit denen für die Komponenten auf der CD-ROM übereinstimmen.

Ich empfehle Ihnen, einige Teile der Typbibliothek genauer anzusehen. Die GUID gibt den Typbibliotheksbezeichner an. `_gtpComp` ist der Name der Hautpschnittstelle für die Klasse. Die Klasse ist eine automatisierungskompatible Schnittstelle, was Sie aus der Tatsache erkennen, daß das `oleautomation`-Attribut gesetzt ist und Informationen von `IDispatch` erbt. Sie enthält die Routine `A` als Teil der Schnittstelle.

Als nächstes wird eine Referenzkopie der kompilierten DLL angelegt. Das ist Version 1. Zeigen Sie die Registerkarte KOMPONENTEN im Dialogfeld PROJEKTEIGENSCHAFTEN an und setzen Sie die Binär-Kompatibilität auf die Version 1 Referenz.

Machen Sie weiter und kompilieren Sie das Projekt so oft Sie wollen. Die Typbibliothek bleibt immer dieselbe. Das bedeutet, Sie können den Code sicher ändern, ohne daß das Einfluß auf die Typbibliotheksinformation hätte. Sie sollten natürlich sicherstellen, daß die Codeänderungen kompatibel zu früheren Versionen sind. Sie sollten die Versionsnummer in der Ressource inkrementieren, so daß die Installationsprogramme wissen, daß sie frühere Versionen Ihrer Komponente durch die verbesserten Versionen ersetzen sollen.

Ändern Sie jetzt den Klassencode, indem Sie eine neue Prozedur B hinzufügen. Deregistrieren Sie die aktuelle DLL mit dem Befehl `regsvr32` unter Angabe der Option `/u`. Damit wird die Registrierung gelöscht, und wir wissen, wir beginnen ganz von vorne mit dieser Applikation. Jetzt kompilieren Sie und betrachten wieder die Typbibliothek:

```
// Generated .IDL file (by the OLE/COM Object Viewer)
//
// typelib filename: gtpComp.dll

[
  uuid(C7C07A2F-23DF-11D2-A754-00001C1AD1F8),
  version(1.1)
]
library gtpCompatibility
{
    // TLib :     // TLib : OLE Automation : {00020430-0000-0000-C000-000000000046}
    importlib("STDOLE2.TLB");

    // Forward declare all types defined in this typelib
    interface _gtpComp;

    [
      odl,
      uuid(C7C07A36-23DF-11D2-A754-00001C1AD1F8),
      version(1.1),
      hidden,
      dual,
```

```
      nonextensible,
      oleautomation
    ]
    interface _gtpComp : IDispatch {
        [id(0x60030000)]
        HRESULT A([in, out] short* Param);
        [id(0x60030001)]
        HRESULT B([in, out] short* Param);
    };

    [
      uuid(C7C07A31-23DF-11D2-A754-00001C1AD1F8),
      version(1.1)
    ]
    coclass gtpComp {
        [default] interface _gtpComp;
    };

    typedef [uuid(C7C07A30-23DF-11D2-A754-00001C1AD1F8), version(1.0), public]
    _gtpComp gtpComp___v0;
};
```

Die Typbibliotheksversion wurde auf Version 1.1 hochgezählt. Ich möchte hier auf einen wichtigen Aspekt aufmerksam machen: Die Versionsnummer der Typbibliothek steht in keinerlei Verhältnis zu der Versionsnummer, die in der Versionsressource angegeben ist. Trotz der Änderung der Typbibliotheksversionsnummer bleibt die GUID der Typbibliothek dieselbe. Die `_gtpComp`-Schnittstelle hat jedoch einen neuen Schnittstellenbezeichner. Wie können aber existierende Versionen des Programms die Schnittstelle nutzen?

Die Antwort ist, daß beide Schnittstellen beibehalten werden. Wenn Sie in der Registrierung nach den Schnittstellen suchen, finden Sie für beide einen Eintrag im Schlüssel `HKEY_CLASSES_ROOT/Interfaces`. Die Originalversion 1.0 der Schnittstelle ist wie folgt definiert:

```
[HKEY_CLASSES_ROOT\Interface\{C7C07A30-23DF-11D2-A754-00001C1AD1F8}]
@="gtpComp"

…\Forward] @="{C7C07A36-23DF-11D2-A754-00001C1AD1F8}"

…\ProxyStubClsid]@="{00020424-0000-0000-C000-000000000046}"

…\ProxyStubClsid32] @="{00020424-0000-0000-C000-000000000046}"
```

Der vollständige CLSID-Eintrag wurde bis auf den ersten weggelassen, um das Ganze zu verdeutlichen. Diese Schnittstelle hat einen Eintrag mit der Markierung Forward, der angibt, daß es eine aktualisierte Version der Schnittstelle gibt. Die Schnittstelle ist hier gezeigt:

```
REGEDIT4

[HKEY_CLASSES_ROOT\Interface\{C7C07A36-23DF-11D2-A754-00001C1AD1F8}]
@="gtpComp"

…\ProxyStubClsid] @="{00020424-0000-0000-C000-000000000046}"

…\ProxyStubClsid32]@="{00020424-0000-0000-C000-000000000046}"

…\TypeLib]@="{C7C07A2F-23DF-11D2-A754-00001C1AD1F8}"
"Version"="1.1"
```

Applikationen, die für die frühere Komponente kompiliert wurden, können die Originalschnittstelle `_gtpComp` anfordern und dann weiterhin korrekt arbeiten. Neuere Applikationen erhalten immer die neuere Schnittstelle.

Wenn Sie noch einmal das Listing für die Typbibliothek betrachten, sehen Sie, daß die neue Routine am Ende der Prozedurliste für die Schnittstelle hinzugefügt wurde. Das bedeutet aus der Sicht der Codierung, daß eine Applikation, die die Schnittstelle nutzt, voll kompatibel mit der neuen ist, auch wenn die frühe Bindung genutzt wird, um diese Funktionen direkt aufzurufen. Ältere Applikationen sehen die neue B-Funktion einfach nicht.

Es gibt zwei zusätzliche Punkte, die Sie aus diesem Beispiel lernen:

- Immer wenn Sie einen neuen Release erzeugen, der eine Schnittstelle neu definiert, erhöhen Sie den Overhead für Ihre Applikation, weil Visual Basic die zusätzliche Typbibliotheksinformation speichern und beide Versionen der Schnittstelle registrieren muß.
- Wenn Sie versuchen, eine neue Applikation mit einer alten Komponente auszuführen, können alle möglichen Probleme auftreten. Das wahrscheinlichste ist, daß Visual Basic einen Laufzeitfehler aufwirft, wenn es feststellt, daß die gewünschte Schnittstelle fehlt. Andre Laufzeitfehler treten auf, wenn Sie die späte Bindung verwenden. Das ist das Problem, das Sie im Abschnitt über Versionsressourcen am Anfang des Kapitels gesehen haben.

Die Abfolge zur Entwicklung aktualisierter Komponenten, die hier beschrieben wurde, zeigt genau, wie Sie in Ihren eigenen Applikationen vorgehen sollten. Für ein ganz neu entwickeltes Projekt wird »Keine Kompatibilität« angegeben. Wenn Sie die Applikation zum ersten Mal kompilieren, sollten Sie Projekt-Kompatibilität auswählen und das Projekt kompatibel mit der neu kompilierten Datei setzen. Damit ist es Ihnen möglich, die Typbibliotheksbezeichner wiederzuverwenden. Wenn Sie die Komponente für die Verwendung durch andere Applikationen freigeben, sollten Sie sie kompilieren und irgendwo eine Referenzkopie anlegen. Sie sollten Binär-Kompatibilität auswählen und sie auf die Referenzkopie setzen, nicht auf das Original.

Jetzt können Sie die Komponente so oft neu kompilieren, wie Sie möchten, und sich darauf verlassen, daß die Typbibliothek und die Schnittstelleninformation kompatibel mit der Referenzkopie bleiben. Sie werden nicht mehrere Schnittstellenversionen in Ihrer Komponente ansammeln, weil Sie die Kompatibilität auf der Referenz basieren lassen. Angenommen, die Referenzversion ist die Typbibliothek Version 1.0. Anschließend erzeugen Sie Version 1.1. Wenn Sie die Binär-Kompatibilität auf der neu erzeugten Komponente aufbauen, wäre die nächste Kompilierung Version 1.2. Weil sie jedoch auf der Referenzkopie basiert, ist Ihre nächste Kompilierung wieder 1.1.

Versuchen Sie jetzt, die Deklaration der Prozedur `A` in der Klasse `gtpComp` zu ändern, um den Parameter zu entfernen, beispielsweise:

```
Public Sub A(param As Integer)
        MsgBox "A Called"
    End Sub
```

Wenn Sie versuchen, das Projekt zu kompilieren, warnt Visual Basic Sie, daß Sie die Abwärtskompatibilität Ihrer Komponente verlieren. Machen Sie es also nicht.

25.2.2 Neue Schnittstellen entwickeln

Warum schafft sich Visual Basic den Aufwand, einen neuen Schnittstellenbezeichner für die aktualisierte Schnittstelle zu definieren, auch wenn diese funktional kompatibel ist? Um die Antwort zu verstehen, gehen Sie noch einmal zurück zur Beschreibung der Komponentenobjektmodelle in den Kapiteln 3 und 4. Eine Schnittstelle wurde als »Vertrag« definiert. Nachdem Sie eine Schnittstelle mit einer bestimmten Menge von Eigenschaften, Methoden und ihren Parameter definiert haben, dürfen Sie sie nicht mehr ändern.

Wenn Sie einer Schnittstelle Eigenschaften oder Methoden hinzufügen, legen Sie letztlich eine neue Schnittstelle an. Sie kann kompatibel sein, wenn sie die alte voll unterstützt. Sie kann auch denselben Namen haben, aber es handelt sich nichtsdestotrotz um eine andere Schnittstelle, die einen eindeutigen Schnittstellenbezeichner hat.

Damit sei die Bedeutung der Beibehaltung einer Abwärtskompatibilität betont. Wenn Sie erlauben, daß sich ein Element einer Schnittstelle oder ihre Parameter ändern, dann zerstören Sie nicht nur die Abwärtskompatibilität in Ihrer Komponente, sondern verletzen auch eine der Hauptregeln, auf der COM basiert.

Das bringt uns zu einem anderen Ansatz, die Abwärtskompatibilität zu bewahren. Statt es Visual Basic zu ermöglichen, neue Schnittstellenbezeichner unter Verwendung der Binär-Kompatibilität hinzuzufügen, erzeugen Sie Ihre eigenen, geänderten Schnittstellen unter Verwendung der Implements-Anweisung. Diese Situation wurde in Kapitel 4 beschrieben, das das Beispiel der `IViewObject`-Schnittstelle vorstellte, die von Containern genutzt wird, um ActiveX-Objekte

anzuzeigen. Als Microsoft der Schnittstelle neue Funktionalität hinzufügen wollte, definierte man eine neue Schnittstelle, `IviewObject2`.

Sie können auch diesen Ansatz verfolgen, aber ändern Sie keine der Schnittstellen, die Sie in anderen Klassen mit der Implements-Anweisung verwenden. Selbst bei der Binär-Kompatibilität verändert die Änderung der Basisklassenelemente den Schnittstellenbezeichner und führt zu Problemen mit anderen Objekten, die die Implements-Anweisung nutzen, um diese Schnittstelle einzubinden. Die Binär-Kompatibilität funktioniert nur mit der Standardschnittstelle einer Klasse. Sie erkennt nicht, ob sich eine implementierte Schnittstelle geändert hat.

25.2.3 Fazit

Die Visual-Basic-Dokumentation ist für die Versionskompatibilität relativ ausführlich, und ich empfehle Ihnen, sie zu lesen. Unter anderem gibt es dort furchterregende Beschreibungen, was passieren kann, wenn Sie Versionsbäume zulassen, wobei Sie Versionen Ihrer Komponente auf mehr als einem Referenzpunkt basieren lassen.

In diesem Kapitel haben Sie gelernt, wie die Versionskompatibilität intern funktioniert, und was in der Typbibliothek und Registrierung passiert, wenn Sie Komponenten aktualisieren. Insbesondere möchte ich dabei zwei Aspekte betonen.

Erstens, die Schnittstellenkompatibilität ist wirklich wichtig, aber vergessen Sie nie die funktionale Kompatibilität. In gewisser Hinsicht verdient sie sogar mehr Aufmerksamkeit als die Schnittstellenkompatibilität, weil Visual Basic keine Hilfe und auch keine anderen Werkzeuge bereitstellt, die Ihnen helfen könnten sicherzustellen, daß Ihre Komponenten abwärtskompatibel mit früheren Versionen sind. Das ist kein Fehler in Visual Basic; es liegt an Ihrem Code, und es gibt keine Sprache, die in dieser Hinsicht besser wäre.

Zweitens sollten Sie die Veröffentlichung nicht vergessen. Die zuverlässige Ausführung Ihrer komponentenbasierte Applikation bedeutet mehr, als nur dafür zu sorgen, daß sie korrekt installiert wird. Sie brauchen vielmehr einen Mechanismus, der erkennt, wenn selbst nach der Installation Komponentenprobleme und Konflikte auf dem Zielsystem auftreten. Und Sie brauchen eine Strategie, wie Sie mit diesen Situationen zurechtkommen können.