# PropertyBag ミステリー

Daniel Appleman 著　文化オリエント株式会社　阿部美奈子訳




今回のストーリーは来日特別編，“ ハイテク探偵 ”が VB の謎に迫ります． なお，登場人物や場所の名前には実在するものもありますが，この記事で起こっている事件は架空のものです．

## 事件のはじまり

天気の長期予報がはじめて当たった， エルニーニョ現象だ．おかげで一日おきに嵐がやってくる．シリコンバレー名物 「中身たっぷりの砂袋」が，百万ドルの大邸宅を海に滑り込む寸前でくいとめていた．そんなわけで事件はゆっくりと進んでゆく． ハイテク専門の私立探偵など， たいして面白いものではない．それにひきかえ連邦第一審裁判所の特別検察官になっていたら…

そのE-mail は夜中の１時に届いた． 私の一日の仕事時間がちょうど半分を過ぎた頃だ．先週閉幕した長野冬季オリンピックの熱狂的な生中継を見ていると，マシンから “ 新着メールがあります ”というぶっきらぼうなメッセージが聞こえてきた． この時間からすると，どこか遠くから来たメールに違いない． ざっと目を通した瞬間にその日の残業が確定した．

メールの主は 「VB で書いたコントロールでサブオブジェクトを使いたい，解決するなら費用がいくらかかってもよい」という．彼が求めている “ サブオブジェクト ”とはいったい何だろう？

シンプルにゆこう．VB5 を開き，空のユーザーコントロール（UserControlプロジェクト）を作成する．そして「ActiveX コントロールインターフェイスウィザード」に従って，Font プロパティをひとつだけ追加する（他のプロパティは消してしまうこと）．Fontプロパティをユーザーコントロールへマップした時点で， コードはリスト1 のようになる．

このコントロールを空のフォームに貼り付ける．するとコントロールのFontプロパティがVBのプロパティウィンドウに表示され，専用の Font プロパティページをもつようになる． ここでフォームを保存してから， テキストエディタを使って .FRM ファイルを開いてみよう． リスト2 のような内容が表示されるはずだ．

非常に興味深い．Fontオブジェクトはフォントの名前・サイズ・属性などを定めるフィールドをいくつかもっているが，これらを個別にSavePropertiesイベントへ保存する必要はな

**リスト 1 ：VB のウィザードが生成したコード**

```
Option Explicit

' 警告！以下のコメント行を変更または削除しないでください！
' マッピング情報 =UserControl,UserControl,-1,Font
Public Property Get Font() As Font
  Set Font = UserControl.Font
End Property

Public Property Set Font(ByVal New_Font As Font)
  Set UserControl.Font = New_Font
  PropertyChanged "Font"
End Property

' ユーザーコントロールのプロパティを初期化します
Private Sub UserControl_InitProperties()
  Set Font = Ambient.Font
End Sub

' 記憶領域からプロパティ値を読み込みます
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
  Set Font = PropBag.ReadProperty("Font", Ambient.Font)
End Sub

' 記憶領域にプロパティ値を書き込みます
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
  Call PropBag.WriteProperty("Font", Font, Ambient.Font)
End Sub
```

**リスト 2**

```
VERSION 5.00
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   3195
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   4680
   LinkTopic       =   "Form1"
   ScaleHeight     =   3195
   ScaleWidth      =   4680
   StartUpPosition =   3       'Windows のデフォルト
   Begin Project1.UserControl1 UserControl11
      Height          =   645
      Left            =   1170
      TabIndex        =   0
      Top             =   720
      Width           =   915
      _ExtentX        =   1614
      _ExtentY        =   1138
   BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   400
      Underline       =   0       'False
      Italic          =   0       'False
      Strikethrough   =   0       'False
   EndProperty
   End
End
```

い．利口なFontオブジェクトは，自らを単一のオペレーションに保存するすべを知っているのだ．上記のリストを見るとフォントの属性は「BeginProperty」と「EndProperty」でくくられ，独立したセクションを形成している．

これがコントロールの最高にクールな特徴のひとつだ．フォントもこれを利用している．ピクチャーや内在するコントロールもまたしかり．唯一あなただけがこの特徴を利用できない．もちろんクラスモジュールを使えば，オブジェクトをコントロールとDLLの中にわけて作成することは可能だ．まにあわせ程度に作成しておき，細かい部分は後からプロパティページで編集すればよい．だが，それらを単一オペレーションのPropertyBagに保存することは不可能だ．つまり，VBで作ったオブジェクトは自己保存できないのである．マイクロソフトはわざわざこの機能をVBから切り離している．

私は依頼主がこの問題を見抜いているかどうかを確かめようと，アップルマン氏が書いたActiveXの解説書，『Visual Basic 5.0プログラマのための詳解ActiveXコンポーネント』（ソフトバンク刊）のページをめくった．しかし関連する記述は見当たらない．PropertyBagをクラスに渡す方法とコントロール名を添えて一連のプロパティを保存する方法は説明してあるが，どうも結論が釈然としない．アップルマン氏は「これに関するより詳しい解説を提供する」と本の中で約束しているが，彼のWebサイト（http://www.desaware.com/desaware）を見ても何も解説されていない．

雨のつたう窓を見つめる．何処から手をつけよう？そのときメール自体に隠された解決への糸口に気づいた．費用はいくらかかってもよい．私はすかさず受話器を取りあげ，24時間後には日本へと向かっていた．

## 答えは東に…

単に嵐を逃れようとしたわけではない．実際この季節の日本はカリフォルニアよりはるかに寒く，ジメジメしている．依頼人が日本におり，費用には限度がない．これだけで日本へ行く理由の95%ができあがる．私は日本へ“精密さ”を求めて旅立ったのだ．自己保存のできるオブジェクトを作成する手段を見つけ出すために極秘調査を行なう必要があると考えていた．

Visual Basicのアスファルトを取り払い，内部オペレーションという下水溝の中にまで潜り込む．さらにコンポーネントオブジェクトモデル（COM）やさまざまなインターフェイスにも調査の手を広げなければならない．インターフェイスは非常に精密な動きをする．メソッドやプロパティの設定，それぞれのパラメータもまた非常に精密なものだ．

日本の超特急“新幹線”をご存知だろうか．この列車は時速600～800kmで走り，3分間隔で次々と発車する．こんな緻密なシステムで列車を走らせることのできる人々であれば，精密なものについて得意に違いない．

日本到着の翌朝，私は秋葉原での調査を開始した．秋葉原――シリコンバレーから来たハイテク“オタク”にとって，そこは天国に一番近い場所といえる．小さな小さな店が軒をならべ，それぞれ特徴を押し出した製品を売っている．電気製品のデパートも立ち並んでいる――幅6メートル，奥行き12メートルの50階建て．各フロアにはそれぞれ特色あるハイテク製品が陳列されている．私はストレージインターフェイスを専門に売るCOMの店をめざし，35階へ直行した．VB内部構造を扱うコーナーには初老の紳士がひとり立っている．彼はかたことの英語さえ話せないにも関わらず，我々の心は完璧に通じ合った･･･

## 老紳士の話

…フォームのプロパティを.FRM ファイルへ保存する場合には，次の 3 つのオブジェクトを扱わなくてはならん．

・Visual Basic （コンテナ）
・フォーム，フォーム上のコントロール，
　プロパティ関連のオブジェクト
・フォームファイル

Visual Basicはフォームにプロパティを保存するように命じ，フォームにフォームファイルオブジェクトへの参照を渡す．そこでフォームはすべてのプロパティを保存して，各コントロールにそれぞれのプロパティをフォームファイルに保存するよう要求する．

この時点ではVB で書いたコントロールは，自分自身の SaveProperties イベントをトリガすると予想できる．フォームオブジェクトはPropertyBagオブジェクトの形式でコントロールへと渡され，その結果コントロールは PropertyBag オブジェクトのWritePropertiesメソッドを使ってプロパティやサブオブジェクトを保存するというわけだな…

この処理の流れは私も知っていた．しかし，さらに詳しく知る必要があったのである．これらのオブジェクトは私が最も目をつけている容疑者たちだ．だから彼らが事件の中でどんな役割を演じているか正確につかむため，それらを完璧に理解しなければならないのだ．

### コンテナ

Visual Basic のフォームはすなわち OCX コンテナだ．ActiveX コントロールを管理し，それらにプロパティを自己保存するよう命じる．現時点ではコンテナとの情報伝達については心配する必要はない．コンテナの仕事はフォームファイルやコントロールへの参照を保つこと，コントロールにフォームへプロパティを自己保存するタイミングを通知することである．

### フォームファイル

コンテナはプロパティをどこかに保存しなくてはならない．通常フォームを保存する場合には，フォームファイルへと保存される．「.FRM」という拡張子の付いたファイルはプロパティに関する記述をテキスト形式でもっている．テキストとして保存できないプロパティは「.FRX」という拡張子が付けられて，バイナリ形式で保存される．これ以外のコンポーネントにはさらに別の拡張子（コントロールならばctlや.ctx）が付けられる．しかし基本はどんな場合でも同じだ．

VBで作成したコントロールはプロパティを保存したファイルをPropertyBagオブジェクトとみなすが，この手のファイルは非常にハイレベルなオブジェクトであり，その動きのほとんどが隠されている．つまり，水面下で「IProperty Bag という名のインターフェイスを起動するオブジェクト」としての役割をもっているわけなのだ．

### コントロール

コントロールは，コントロールのプロパティを保存する際に発生するSave Propertiesというイベントをもっている．このイベントの機能は，ひとつの点を除いて明らかだ．PropBag.WritePropertyメソッドをFont オブジェクトや Picture オブジェクトなどのサブオブジェクトのために呼び出すと，PropertyBagオブジェクトはなんとかしてそのサブオブジェクトを保存する方法を探ろうとする．一方このメソッドをクラスモジュールを用いて作成したオブジェクトのために呼び出すと，エラーが発生してしまう．そして次には…

### サブオブジェクト

Font オブジェクトと Picture オブジェクトの作成したオブジェクトが異なる理由は何だろう？ その答えは難解であると同時に単純だ．

Fontオブジェクトのメインインターフェイスがフォント情報へアクセスするためのプロパティを含んでいることは周知のとおりだ．だが，Font オブジェクトがそれ以外に付加的なインターフェイスをもっているということはあまり知られていない．COM コンポーネントは必要になると，インターフェイスをいくつでもサポートでき，Font オブジェクトは少なくとも上記のもの以外に3 つのインターフェイスをサポートできる．

このうち我々が取り上げているのは「IPersistPropertyBag」というインターフェイスだ．オブジェクトがいったんこのインターフェイスを実装すると，プロパティをPropertyBagに保存できるようになったことが明示的になる．

## 犯罪計画

老紳士の話に私はぴんときた．まるでこんな会話を見ているかのように，すべてのつじつまが合ってゆく…

**Visual Basic：**「開発者」はフォームを保存したいと望まれている．そこで私はテキストファイルを開いて，IPropertyBag インターフェイスを実装するオブジェクトをひとつ作成した．IPropertyBag インターフェイスの Write メソッドをプロパティのために呼び出せば，そのインターフェイスは可能なかぎりテキストに変換されてフォームファイルに保存され，バイナリデータがあれば .FRX ファイルへと送られるのが常なのだ… しかしこれは Visual Basic での話であり，IPropertyBag インターフェイスが Visual Basic と直接互換性があるわけではない．

私はVBから呼び出し可能なWritePropertyメソッドをもつ専用のPropertyBagオブジェクトを作ろうと思っている．さあコントロール君，君がプロパティを保存する番だ．文句はいわないでくれ．PropertyBagオブジェクトへプロパティを書き込むのだ．逆らえば君の身の保証はないと思ってほしい．

**コントロール：**わかりました．おっしゃる通りにいたします．こ，これがすでに書き込まれた私の1番目のプロパティ，そしてこちらがFontオブジェクトでございます．しかし，Fontオブジェクトのプロパティをどのように保存したらよいのかは存じあげません．どうか命だけは…

**Visual Basic：**ふむ，気の効かないオブジェクトだ．取り調べでもしてみよう．きみ，Fontオブジェクト君．ここにQueryInterfaceというオペレーションが来ていて，君が私と話ができるかどうかを確かめたいと言っている．君はIPersistPropertyBagインターフェイスをサポートしているのか？

**Fontオブジェクト：**これが俺がもってるIpersistPropertyBagインターフェイスへのポインタさ．どうにでもしてくれ．

**Visual Basic：**ならば君のIPersist PropertyBagインターフェイス上でSaveメソッドを呼び出すとしよう．パラメータのひとつが私のIPropertyBagインターフェイスへのポインタとなるのだ．これでどうかね？

**Font オブジェクト：**へえ，そうかい．じゃあお前さんのIPropertyBagインターフェイス上で，俺のプロパティのためにWirteメソッドを呼び出すとしようじゃねぇか．それならお前さんもそいつらをサブオブジェクトとして保存できるだろうよ．

**Visual Basic：**コントロール君．よくやった．私はFontオブジェクトと話ができたし，Fontオブジェクトも自分のプロパティを保存できたようだ．まだ何か言いたいことがあるかね？

**コントロール：**いいえ，ございません．すべてのプロパティが保存できました．私はこれでおいとまさせて頂きます．

老紳士はオブジェクト階層と，各オブジェクトに属するインターフェイスを示した一覧を無造作に書き上げた（図1）．そのとき私は突然胃の中にこみあげる何かを感じた．老紳士は微笑みながら角のドアを指差したが，私は「違う．そんなことじゃない」と言って，もうひとつのインターフェイスを指差した．IStreamとは？IPersistStreamとは？

彼はフォームファイルを指差し，私はそれを理解した．PropertyBagはプロパティをテキスト形式で保存するように設計されている．そのため，これらのプロパティは人間が読んでも簡単に理解でき，テキストエディタによる編集も可能なのだ．これはプロパティをフォームファイルへと保存するときに有効だが，コントロールがファイルに保存されてない状態でプロパティを保存しなければならない場合も多い．たとえばVBのIDE上でプログラムをインタプリタ実行する場合，作成した各コントロール（のインスタンス)はそのつど破棄され，また再生される．ただしこれは実行時の話だ．

デザイン時のコントロールのプロパティは，コントロールが破棄される前に保存されなければならない．そのため新しく作成された実行時のコントロールによってロードできるようになっている．この場合，プロパティをディスク上のファイルへ保存するのがどれだけ時間のムダか想像できるだろう．実際プロパティの値をテキストに変換するというのも時間のムダだ．だからVisual Basicはそれぞれの状況に合った手段を取るのである．

Visual BasicはOLE構造化ストレージを用いて内部的にフォーマットされたメモリブロックの中へプロパティを保存する．これは非常に複雑なドキュメントを個々のデータストリームから作り上げるためのテクニックだ（ほとんどのVBプログラマは構造化ストレージを直接使った経験がないだろう．ＶＢから直接これを使うのは不可能に近い．しかしDesaware社のStorageToolsを使えばこの種のファイルを

**図1：オブジェクト階層とそのインターフェイス**

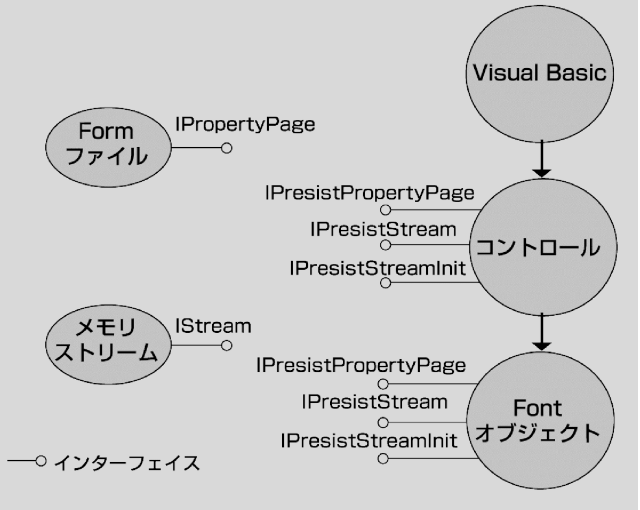
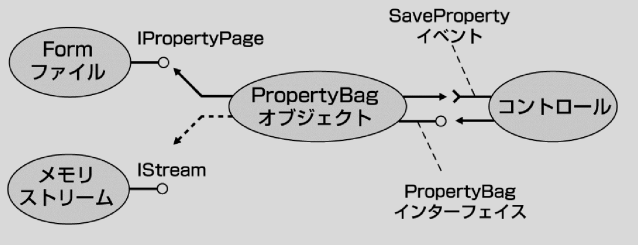



**図2：PropertyBagオブジェクトの動作概念図**



VBから簡単に作成・管理できる).

メモリブロックはIStreamというインターフェイスを実装したオブジェクトを使って参照される．IStreamはバイナリデータを読み書きするメソッドをもつインターフェイスだ． VBはデータをフォームファイルに書き込む際にIPropertyBagインターフェイスを使用し，一方でメモリストリームへ書き込みを行なう際にIStreamインターフェイスを使用する．オブジェクトはプロパティをフォームファイルへ保存できるということを示すためにIPersistPropertyBagインターフェイスを実装している．これとまったく同じく，プロパティをメモリストリームへ保存できることを示すためにIPersistStreamインターフェイスとIPersistStreamInitインターフェイスを実装する．残念ながらこのメカニズムはＶＢでコントロールを書く段階ではPropertyBagオブジェクトによって隠されている．図2からわかるように，PropertyBagオブジェクトのWritePropertyメソッドは，Visual Basicがデータを保存する場所によって，データをフォームファイルにもメモリストリームにも保存できるのだ．

## ロードせよ！

これでプロパティがどのように隠蔽されているのかが理解できた．しかし調査の手がゆるんだあとで窃盗犯が戦利品を取り戻すように，VBからそれらを再ロードできなければプロパティの保存は無用の長物になってしまう．

一目みて明らかなことだが，VBはフォームをロードするときに，コントロールやサブオブジェクトに対して単純にそれぞれのフォームファイルやメモリストリームからプロパティをロードするよう命令する．しかしここに落とし穴がある．

フォームは，コントロールやサブオブジェクトが存在しない場合，どうやってこれらにプロパティのロードを命じるのだろう？ ロードされたばかりの空のフォームにはコントロールはひとつもないではないか．私はこの疑問を老紳士に突きつけた．すると彼はまたも右手にあるドアを指差したので私が首を振ると，彼は気違いのように笑い出した．

私に“もう話しても無駄だ”と悟らせるためなのだろう，無駄に終わった追跡にイライラと彼に背を向けて立ち去ろうとした瞬間，彼が何かを叫ぶのが聞こえた．何を意味しているのかを尋ねようと振り返ったが，彼の姿はもうなかった．

エスカレーターで1階へと降りると，もう夕暮れが迫っていた．地下鉄へ乗ろうと近くの駅へ入った私は，50人乗りの車両に400人もが乗り込む日本人の様子を目のあたりにした．これは未知の世界へと開かれた扉ではないだろうか…

気が付くと小学生らしい数十人の子供が，私を取り囲んでいた．子供たちは私にバレンタインカードを差し出したのだが私は夢と現実の境目にいるような気がしてならなかった．カードの1枚を開けてみると，そこにはこう書いてあるではないか．

**Begin Project1.UserControl1 UserControl11**
**BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}**
**ハッピーバレンタイン…**

まさにそれは私の求めている答えだった．フォームファイルを読み込むときにVisual Basicはオブジェクトのプロパティの開始点を示す2種類のステートメントを読む．ActiveXコントロールといったオブジェクトのプログラミング上の名前を伴なうBeginステートメントと，オブジェクトのGUID（Grobal Unique IDentifier）に続いてオブジェクトの名前を伴なうBeginPropertyステートメントである．

Visual Basicは， 特定のコントロールやオブジェクトをサポートする実行プログラム，あるいはDLLに関する情報をコントロールのプログラミング名やオブジェクトのGUIDを利用してレジストリから検索する．そして検出したプログラムやDLLをロードし，オブジェクトのインスタンスを作成するように命じるのである． インスタンスが用意できると，VBは必要な初期化をすべて実行できるようになる．これにはコントロールやオブジェクトへIPersistPropertyBagやIPersistStreamインターフェイスを要求する命令も含まれている． こうしてVBはPropertyBagかメモリストリームから，コントロールやオブジェクトにプロパティをロードさせることが可能になるのだ．

断片的だったすべてのものがひとつにまとまった． コントロールやサブオブジェクトがどのようにしてそれぞれのプロパティと内部データを保存・ロードできるのかを理解できたのだ．唯一残っているのは，FontオブジェクトのようにVBを使ってサブオブジェクトを作る方法だ．

とりあえず今は，そのようなオブジェクトには次の任務があるということだけが明らかである．

・IPersistPropertyBagインターフェイスの実装
・IPersistStreamインターフェイスの実装
・IPersistStreamInitインターフェイスの実装
・VBが提供するオブジェクト上でIPropertyBagインターフェイスのメソッドを呼び出せるようになる
・VBが提供するオブジェクト上でIStreamインターフェイスのメソッドを呼び出せるようになる
・独自のDLL内にパブリックオブジェクトとして実装される

最後の項目はどういうことだろう？ コントロールがサブオブジェクトをロードする場合，最初に参照するのはオブジェクトのGUIDである．このGUIDはVBがそのオブジェクトの作成方法やサポート方法を知っているEXEかDLLを探し出せるようにするため，システムレジストリ内に保存されなければならない．すなわちオブジェクトはパブリックでなければならないということだ．プライベートなクラスモジュールを使って作成されたオブジェクトは，システムレジストリに記憶されな

**リスト 3**

```
' サブオブジェクトの使用法を示すコントロール
' Copyright (c) 1998 by Desaware Inc. All Rights Reserved

Option Explicit
' プロパティの変数:
Dim m_SubObject As ControlSubObject

' 警告！以下のコメント行を変更または削除しないでください！
' マッピング情報 =UserControl,UserControl,-1,Font
Public Property Get Font() As Font
  Set Font = UserControl.Font
End Property

Public Property Set Font(ByVal New_Font As Font)
  Set UserControl.Font = New_Font
  PropertyChanged "Font"
End Property

' SubObject プロパティは VB のプロパティページで
' 表示されるようになるために， あるトリックを使います．
' このプロパティはデザイン時には文字列型ですが，
' 実行時にはオブジェクト参照型となります．
' 次のプロシージャ属性をプロパティページへマップする
' このプロパティをオブジェクトのプロパティページへ
' 設定します． より洗練された方法は， Desaware 社の
' SpyWorks ActiveX 拡張 DLL を使って IPerPropertyBrowsing
' インターフェイスを実装し， VB のプロパティウィンドウへは
' たとえユーザーが編集しようとしても設計時に指定した
' メッセージを常に強制表示させるという方法です．

Public Property Get SubObject() As Variant
  If Ambient.UserMode Then
    Set SubObject = m_SubObject
  Else
    SubObject = "Sub object"
  End If
End Property

Public Property Set SubObject(ByVal New_SubObject As ControlSubObject)
  Set m_SubObject = New_SubObject
  PropertyChanged "SubObject"
End Property

Public Property Let SubObject(ByVal New_Object As Variant)
  If Ambient.UserMode Then
    Err.Raise 382
  End If
End Property

  ' ユーザーコントロールのプロパティを初期化します．
Private Sub UserControl_InitProperties()
  Set Font = Ambient.Font
  Set m_SubObject = New ControlSubObject
  m_SubObject.Text = "Default Value"
  PropertyChanged "SubObject"
End Sub

' コントロールがサブオブジェクトの中身を表示します．
Private Sub UserControl_Paint()
  CurrentY = 0
  If Not (m_SubObject Is Nothing) Then
    With m_SubObject
      Print .X
      Print .Y
      Print .Text
    End With
  Else
    Print "Empty"
  End If
End Sub

  ' 記憶領域からプロパティ値を読み込みます．
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
  Set Font = PropBag.ReadProperty("Font", Ambient.Font)
  Set m_SubObject = PropBag.ReadProperty("SubObject")
End Sub

  ' 記憶領域にプロパティ値を書き込みます．
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
  Call PropBag.WriteProperty("Font", Font, Ambient.Font)
  Call PropBag.WriteProperty("SubObject", m_SubObject)
End Sub
```

い．VBにおいてActiveXコントロールのプロジェクトはパブリックオブジェクトをサポートできないため，オブジェクトは明示的に別のDLLプロジェクト内へ保存される必要があるのだ．

では，最初の5つの項目はどのようにクリアすればよいだろう？ ここで挙げたインターフェイスはすべて隠蔽されているか，あるいはVisual Basicとの互換性を持ち合わせていない．私がタイプライブラリをいくつか作成したり，システムレジストリをVBから使えるように修正することは理論上可能だった． しかしレジストリをむやみに変更するのは避けたい． システム全体をだいなしにする可能性があまりに高すぎる．

その子供たちは私が考えている間じっと黙っていたが，私が口を開くのを今か今かと待っている．そこで，ふざけて彼らに聞いてみた．「まさか，VBを使って任意のインターフェイスを実装したり呼び出したりする方法を知るわけないよね」．すると彼らは声を合わせて 「Mr.Powerに訊けば？」と，長く急な丘を指差した．

## 峠の邂逅

頂上までは実に長かった．気が付くとたくさんの鯉が泳ぐ池に囲まれた日本の古い寺が目の前にあった． 私は他の旅行者と同様，興味を押さえ切れずに鯉にエサを与えた．

**リスト 4**

```
Private Sub IdwCustomOleHook_GetInterfaceCount(iCount As Long)
  '3 つのカスタムインターフェイスを実装
  iCount = 3
End Sub

Private Sub IdwCustomOleHook_GetInterfaceName(ByVal InterfaceNumber As Long, _
                                              InterfaceName As String)
  'C 言語 IDL ファイルからの GUID
  Select Case InterfaceNumber
    Case 0
      'IPersistPropertyBag
      InterfaceName = "{37D84F60-42CB-11CE-8135-00AA004BB851}"
    Case 1
      'IPersistStreamInit
      InterfaceName = "{7FD52380-4E07-101B-AE2D-08002B2EC713}"
    Case 2
      'IPersistStream
      InterfaceName = "{00000109-0000-0000-C000-000000000046}"
  End Select
End Sub
```

**リスト 5**

```
Private Sub IdwCustomOleHook_GetInterfaceVtbl(ByVal InterfaceNumber As Long, _
                                              FunctionAddresses() As Long)
  Select Case InterfaceNumber
    Case 0
      'IPersistPropertyBag
      ReDim FunctionAddresses(3)
      FunctionAddresses(0) = GetAddress(AddressOf PropBagGetClsid)
      FunctionAddresses(1) = GetAddress(AddressOf PropBagInitNew)
      FunctionAddresses(2) = GetAddress(AddressOf PropBagLoad)
      FunctionAddresses(3) = GetAddress(AddressOf PropBagSave)
    Case 1, 2
      ' IPersistStream とIPersistStreamInit の唯一の違いは
      ' 最後の関数にある （IPersistStream は InitNew メソッドを持たない）
      ' InitNew メソッドを持たないからといって，実際は余分なポインタが
      ' 無視されるだけのことだ．
      ReDim FunctionAddresses(5)
      FunctionAddresses(0) = GetAddress(AddressOf PropBagGetClsid)
      FunctionAddresses(1) = GetAddress(AddressOf PSIsDirty)
      FunctionAddresses(2) = GetAddress(AddressOf PSLoad)
      FunctionAddresses(3) = GetAddress(AddressOf PSSave)
      FunctionAddresses(4) = GetAddress(AddressOf PSGetMaxSize)
      FunctionAddresses(5) = GetAddress(AddressOf PSInitNew)
  End Select
End Sub
```

それから振り返って神聖な寺の中に入るとそこに彼がいた．そう，Mr.Powerだ．彼はビルマの竪琴と曼荼羅をたずさえて座っていた．私はサブオブジェクトで進退極まった状況を話そうとしたが，彼は横に座るように手招きし，それから3時間というもの問題を解決する糸口を聞きだそうとあせる私を尻目に，忍耐強く竪琴の弾き方を教えたのである．

どうにか私は，とちりながらもラフマニノフの交響曲3番を竪琴で最後まで演奏した．すると彼は立ち上がってお辞儀をし，セッションは終わりだと告げるではないか．私が失意に首を振りながら立ち去ろうとすると突然彼が口笛を吹いた．振り向くと彼は，ひとつの単語を口にした． その瞬間，稲妻のように私の心にある考えがひらめいた．ついに答えが出たのだ．

## 解答はいずこ？

Mr. Power が私に与えてくれた言葉は「SpyWorks」だった． なぜ思い付かなかったのだろう． サンプルコードが Desaware 社の Web サイト（ftp://ftp.desaware.com/SampleCode/Articles/subobject.zip）で入手可能だ．このサンプルコードは「ControlSubObjects」「SubObjectComponent」の 2 つのオブジェクトからなり，「SpyWorks」のプロフェッショナル版を使って実行できるものだが，それ以外の多くの人にも利用できるものだろう．これらの原理は ATL や MFC を使用しているVC++プログラマにも応用できるものだ．

リスト3はControlSubObjectsコントロールが，新しく作成したControlSubObject オブジェクトをFontオブジェクト同様に扱えるということを示すリストの一部である．

ご覧の通り，新しい ControlSubObject オブジェクトは完全に自己保存が可能である． 本当のマジックはSubObjectComponentにおけるオブジェクトの実装に隠されているのだが， ここではPropertyBag の実装に的を絞ろう． メモリストリームのインプリメントも， これとほぼ同じである．

プロジェクトには 「ControlSubObject」というクラスモジュールがある．最初のステップはこのクラスをIPersistPropertyBag インターフェイスにインプリメントすることだ．この処理は，SpyWorks に含まれている ActiveX 拡張ライブラリを参照設定へ追加することで完了する． こうしてdwControlHookオブジェクトというオブジェクトの作成とIdwCustomOleHookというインターフェイスの実装が可能になった．dwControlHook オブジェクトはControlHook と呼ばれ，クラス初期化ルーチン内で初期化される． Interfaces と名付けた配列は各インターフェイス用のIID 値を保持するように定義される．各標準インターフェイスが固有のインターフェイス識別子を持っているのはご承知の通りだ．標準インターフェイスのこの値はレジストリ （VC++ の場合は .IDL ファイル）検索により取得できる．

**リスト 6**

```
' IPersistPropertyBag の Load メソッド
Public Function PropBagLoad(ByVal obj As IUnknown, _
        ByVal p As IUnknown, ByVal IErrorLog As IUnknown) As Long
  Dim myobj As ControlSubObject
  Set myobj = obj
  Call myobj.intPropBagLoad(p, IErrorLog)
End Function

' IPersistPropertyBag の Save メソッド
Public Function PropBagSave(ByVal obj As IUnknown, _
        ByVal p As IUnknown, ByVal fClearDirty As Long, _
        ByVal fSaveAllProperties As Long) As Long
  Dim myobj As ControlSubObject
  Set myobj = obj
  Call myobj.intPropBagSave(p)
End Function
```

```
' サブオブジェクトのサンプルコンポーネント
' Copyright (c) 1998 by Desaware Inc.
' All Rights Reserved

Option Explicit

Dim ControlHook As dwControlHook
Implements IdwCustomOleHook
' 3 つのインターフェイス IID 用スペース
Dim Interfaces(15, 2) As Byte

Private Sub Class_Initialize()
  Set ControlHook = New dwControlHook
  ControlHook.Initialize Me
End Sub
```

VB の Implements ステートメントを使って実装された IdwCustomOleHook インターフェイスは，ActiveX 拡張 DLL に実装したいインターフェイスのリストを初期化するために使用する．このサンプルではIdwCustomOleHook_GetInterfaceNameメソッドが呼び出されるときに各標準インターフェイスのIIDをInterfaceNameパラメータ内へ設定する（リスト4）．単に「InterfaceName ="IPersistPropertyBag"」と指定すればレジストリの検索は可能だが，私はどちらかというと実際のIID 文字列を使用したい．もしレジストリが標準インターフェイスの識別子を見逃しでもしたら大変だし，この方が速いという利点もある．

加えてインターフェイスのメソッドをインプリメントするために使用する関数を，ActiveX 拡張 DLL に通知する必要もある．これらの関数は標準モジュール内にあるべきものなので，それらのアドレスは AddressOf 演算子を用いて取得すればよい（リスト5）．

ControlMethodsと名付けられた標準モジュールを調べると，IPersistPropertyBagインターフェイスのLoadメソッドとSaveメソッドをインプリメントするリスト6 の関数が見つかるはずだ．各関数の最初のパラメータは，それ自体がオブジェクトである．そのため関数呼び出し時にどのオブジェクトが参照されているかを簡単に把握できる．この場合「obj」パラメータに，フレンド関数を呼び出す正しいオブジェクトを割り当てる．このインターフェイスメカニズムの内部処理はコンポーネントに対しプライベートな状態を保っている．2つめのパラメータ「p」はプロパティが保存されるオブジェクト（典型的な例がフォームファイル）への参照だ．このオブジェクト参照はIPropertyBag インターフェイスへのポインタであり，実際にプロパティの保存やロードを実行するクラスメソッドへと戻される．

クラスモジュールを振り返ってみると，LoadメソッドもSaveメソッドもIPropertyBag を参照する「p」パラメータと我々がほとんど無視するエラーログパラメータを受け取る．ではどうやってVBと互換性を持たないIPropertyBagインターフェイス上でメソッドを呼び出すというのだろうか？ その答えは，SpyWorks のジェネリックコールである．2 つのエントリポイントが，それらの関数のためにVB標準の標準的なDeclareステートメントを使って定義される．

```
Private Declare Function PropertyBagRead _
 Lib "dwaxextn.dll" Alias "dwGenericCall" _
 (ByVal ObjectReference As Long, _
 ByVal s As Long, v As Variant, _
 ByVal IErrorLog As IUnknown) As Long

Private Declare Function PropertyBagWrite _
 Lib "dwaxextn.dll" Alias "dwGenericCall" _
 (ByVal ObjectReference As Long, _
  ByVal s As Long, v As Variant) As Long
```

このコードを見て混乱されるかもしれない．経験を積んだVBプログラマならば，Declareステートメントがエクスポートされた関数の呼び出しに使用されるものということを知っているはずだからだ．つまりDeclare ステートメントはオブジェクトに属したメソッドの呼び出しには使えないのである．それにAlias ステートメントのおかげで，これら2 つの Declare ステートメントが実はまったく同じ関数を呼び出すだけということにも気づかされるだろう．さらにそれらは数と種類の異なるパラメータをもっているのだ．複数の，しかも異なる種類のパラメータをもっている同じ名前の関数など一体誰が知っているものか．

**リスト 7**

```
'IPersistPropertyBag.Load メソッドのインプリメント
Friend Function intPropBagLoad(p As IUnknown, elog As IUnknown) As Long
  Dim gencall As New dwGenericCall
  Dim v As Variant
  Dim pname$
  Dim hres As Long
  ' IPropertyBag を使用するための gencall を設定します.
  Call gencall.SetInterfaceInfo("{55272A00-42CB-11CE-8135-00AA004BB851}", p)
  ' 各サブプロパティに渡すものは
  ' 1 :プロパティ名としてのヌル文字終了 Unicode 文字列
  ' 2 :ロードするデータの型へ設定されるバリアント値
  ' 3 :Load 関数の呼び出しにより提供されるエラーログ用 IErrorInfo オブジェクト
  pname = "X" & Chr$(0)
  hres = PropertyBagRead(gencall.GenericCallReference(3),StrPtr(pname), v, elog)
  m_x = v
  pname = "Y" & Chr$(0)
  hres = PropertyBagRead(gencall.GenericCallReference(3),StrPtr(pname), v, elog)
  m_y = v
  pname = "Text" & Chr$(0)
  v = ""   ' 文字列型に設定します.
  hres = PropertyBagRead(gencall.GenericCallReference(3),StrPtr(pname), v, elog)
  m_text = v
End Function

' IPersistPropertyBag.Save メソッドのインプリメント
Friend Function intPropBagSave(p As IUnknown) As Long
  Dim gencall As New dwGenericCall
  Dim v As Variant
  Dim s As String
  ' IPropertyBag を使用するための gencall を設定します.
  Call gencall.SetInterfaceInfo("{55272A00-42CB-11CE-8135-00AA004BB851}", p)
  ' 各サブプロパティのために
  ' プロパティを保存用バリアント値にロード
  ' それをプロパティ名を表わすヌル文字終了 Unicode 文字列と一緒に渡します.
  v = m_x
  s = "X" & Chr$(0)
  Call PropertyBagWrite(gencall.GenericCallReference(4), StrPtr(s), v)
  v = m_y
  s = "Y" & Chr$(0)
  Call PropertyBagWrite(gencall.GenericCallReference(4), StrPtr(s), v)
  v = m_text
  s = "Text" & Chr$(0)
  Call PropertyBagWrite(gencall.GenericCallReference(4), StrPtr(s), v)
End Function
```

## あばかれたトリック

これは神秘的な東洋哲学とは違う．マジックでもない．ただ単にVBのDeclareステートメントがもたらすとてつもない柔軟性を活用しながら，インターフェイスのメソッド呼び出しにおける問題を解決してくれる信頼性の高い方法なのである（リスト7）．そのトリックとは――「dwGenericCall」というオブジェクトの作成だ．このオブジェクトを作成したら，どのオブジェクトを使用しているのか，どのインターフェイスをオブジェクトへのアクセスに使いたいのかを通知するためにSetInterfaceInfoメソッドを実行する．この2つを通知するために，データの保存先であるオブジェクトを参照する「p」パラメータを使うのだ．

そしてIPropertyBagインターフェイスをIID文字列で指定し，先ほど宣言しておいたPropertyBagRead関数とPropertyBagWrite関数を実行する．失敗しない唯一のコツは第1パラメータだ．このパラメータはジェネリックオブジェクト上で呼び出したGenericCallReference関数の戻り値を取得するものだが，実は単に呼び出し中のメソッドの位置を指すものだ．たとえばIPropertyBagインターフェイスのLoadメソッドは，インターフェイス内における4番目のメソッドである（各インターフェイスの頭から3つのメソッドは，それすなわちIUnknownインターフェイスに属する3つのメソッド――AddRef，ReleaseそしてQueryInterfaceだ）．この関数を呼び出すと，ActiveX拡張ライブラリが第1パラメータを読み，関数へ渡したその他のパラメータを使って自動的に正しいメソッドを呼び出す．

パラメータの型はDeclareステートメントで定義しているので，パラメータとして何をメソッドへ渡すのかを完全に制御できる．このサンプルの場合，IPropertyBagのLoadメソッドを実行するには“プロパティ名を含んだヌル文字終了のUnicode文字列へのポインタ”が必要である．その文字列をUnicodeバイト配列へと変換し，参照をその配列の第1バイトに渡す．しかし今回のケースでは，ヌル文字終了の文字列を作成し，Visual Basicが内部的に格納している文字列へのポインタを渡す方がずっと簡単だった（VBは文字列を常にUnicodeとして保持している）．

このサンプルコードの動作は，サブオブジェクトを使用したコントロールをもっているフォームファイルのリスト（リスト8）で確認できる．ご覧のとおり，オブジェクトは自らの内部プロパティを保存するだけでなく，期待通りの結果をBeginPropertyステートメントとEndPropertyステートメントの間に表示してくれた．

## エル・ニーニョ再び

誌面にのせたコードはプロジェクトのごく一部である．完全なサンプルプロジェクトにはIPersistStreamインターフェイスとIPersistStreamInitインターフェイスの実装が含まれ，IStreamインターフェイスの呼び出し方法を見ることができる．それに加えてFontオブジェクトやPictureオブジェクトと同様，サブオブジェクトも編集可能にするPropertyBagのインプリメントについても解説している．またプロパティページをよりクリエイティブに活用するための詳しい情報は，『詳解ActiveXコンポーネント』（前出）を参照してほしい．

時差ぼけの回復には実に数週間かかった．きっとそれは日本から戻った2日後にワシントンD.Cへ出向かなければならなかったからだ．それに，日本への旅は有益だったと言いたいところだが，やっとこプラマイゼロである．なぜかって？ 帰国の前に秋葉原へ立ち寄ったのだが，そこにはすばらしいハイテク機器がずらりと並んでいて，思わず財布の紐が…

**リスト 8**

```
VERSION 5.00
Object = "*\AControlSubObjects.vbp"
Begin VB.Form Form1
  Caption            =  "Form1"
  ClientHeight       =  3195
  ClientLeft         =  60
  ClientTop          =  345
  ClientWidth        =  4680
  LinkTopic          =  "Form1"
  ScaleHeight        =  3195
  ScaleWidth         =  4680
  StartUpPosition    =  3  'Windows のデフォルト
  Begin ControlSubObjects.ControlContainsObjects
  ControlContainsObjects1
    Height           =  1365
    Left             =  900
    TabIndex         =  0
    Top              =  360
    Width            =  1275
    _ExtentX         =  2249
    _ExtentY         =  2408
    BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
      Name           =  "MS Sans Serif"
      Size           =  8.25
      Charset        =  0
      Weight         =  400
      Underline      =  0   'False
      Italic         =  0   'False
      Strikethrough  =  0   'False
    EndProperty
    BeginProperty SubObject {7A98FA31-AC29-11D1-B78F-00001C1AD1F8}
      X              =  6
      Y              =  0
      Text           =  "New value2"
    EndProperty
  End
End
```

*‘ The Object that came in from the Code ’*
*by Daniel Appleman.*


Daniel Appleman 氏は，“ the Visual Basic Programmer's Guide to the WindowsAPI ”，“ the Visual Basic Programmer's Guide to the Win32 API ”，および “ How Computer Programming Works ” の著者であり，1985 年に Microsoft Windows がリリースされたとき以来，Windows アプリケーションの開発を続けている．
彼は 1991 年に Desaware Inc. を設立した．これはカリフォルニア州のキャンベルを拠点とするソフトウェア会社で，コンポーネントソフトウェアと開発者向けの上級ツールに焦点を絞っている．
連絡先は: dan@desaware.com
http://www.desaware.com/desaware