ISSN: 1545-679X

Information Systems Education Journal

Volume 3, Number 23 http://isedj.org/3/23/ August 4, 2005

In this issue:

Teaching the Blind to Program Visually

Robert M. Siegfried
Adelphi University
Garden City, NY 11530, USA

Denis Diakoniarakis
Adelphi University
Garden City, NY 11530, USA

Uchechukwu Obianyo-Agu
Adelphi University
Garden City, NY 11530, USA

Abstract: The proliferation of graphical user interfaces has had a dramatic impact on the ability of the blind to work as programmers. It is particularly difficult for the blind to design forms for programs written in Visual Basic. A prototype scripting language is introduced that enables the blind to create Visual Basic forms without the need to specify all the detailed information that Visual Basic requires and without the "point and click" approach that the blind cannot use. The syntax for the language is described and plans for expanding the language are discussed.

Keywords: blind programmer, visually impaired, graphical user interfaces, special education

---

Recommended Citation: Siegfried, Diakoniarakis, and Obianyo-Agu (2005). Teaching the Blind to Program Visually. *Information Systems Education Journal, 3* (23). http://isedj.org/3/23/. ISSN: 1545-679X. (Also appears in *The Proceedings of ISECON 2004:* §3265. ISSN: 1542-7382.)

This issue is on the Internet at **http://isedj.org/3/23/**

The **Information Systems Education Journal** (ISEDJ) is a peer-reviewed academic journal published by the Education Special Interest Group (EDSIG) of the Association of Information Technology Professionals (AITP, Chicago, Illinois). • ISSN: 1545-679X. • First issue: 8 Sep 2003. • Title: Information Systems Education Journal. Variants: IS Education Journal; ISEDJ. • Physical format: online. • Publishing frequency: irregular; as each article is approved, it is published immediately and constitutes a complete separate issue of the current volume. • Single issue price: free. • Subscription address: subscribe@isedj.org. • Subscription price: free. • Electronic access: http://isedj.org/ • Contact person: Don Colton (editor@isedj.org)

EDSIG activities include the publication of ISEDJ, the organization and execution of the annual ISECON conference held each fall, the publication of the Journal of Information Systems Education (JISE), and the designation and honoring of an IS Educator of the Year. • The Foundation for Information Technology Education has been the key sponsor of ISECON over the years. • The Association for Information Technology Professionals (AITP) provides the corporate umbrella under which EDSIG operates.

Teaching the Blind to Program Visually

Robert M. Siegfried
siegfrir@panther.adelphi.edu

Denis Diakoniarakis

Uchechukwu Obianyo-Agu

Department of Mathematics and Computer Science
Adelphi University
Garden City, NY 11530, USA

ABSTRACT

The proliferation of graphical user interfaces has had a dramatic impact on the ability of the blind to work as programmers. It is particularly difficult for the blind to design forms for programs written in Visual Basic. A prototype scripting language is introduced that enables the blind to create Visual Basic forms without the need to specify all the detailed information that Visual Basic requires and without the "point and click" approach that the blind cannot use. The syntax for the language is described and plans for expanding the language are discussed.

Keywords: blind programmer, visually impaired, graphical user interfaces, special education

1. INTRODUCTION

Text-based, interactive computing has existed since the early 1960s (Palfreman 1991). As interactive operating systems, minicomputers, and microcomputers were developed, the command-line interface remained the standard means of communicating with the computer. This began to change with the development of the Xerox Alto (Ceruzzi 2000), which led to GUIs such as X-Windows, the MacIntosh operating system, and Microsoft Windows, which allowed Bill Gates to fulfill the ambition of making it easy for his mother to learn how to use a computer (Wallace 1993).

Computers have also provided an accessible means of employment for the blind and visually impaired. This has been unusual, given that the blind have a very high unemployment rate (Kirchner 1997). This has been facilitated by a collection of specialized tools including screen readers such as JAWS, speech recognition programs such as Dragon, and Braille terminals and screen enlargers for the visually impaired. These tools and the relevant training have allowed the blind to compete in the text-based world of computers.

Since Microsoft Windows version 3 debuted in 1990, graphical user interfaces have become the standard for modern computing. The change has also had a dramatic impact on the job market for blind programmers. According to Janina Sajka, Director of Information Technology for the American Federation for the Blind (AFB), most Windows-based applications are developed using software tools that the blind cannot use (a 2004 private communication by J. Sajka to U. Obianyo-Agu). Additionally, it has been difficult to update screen readers frequently enough to remain current with new developments in GUIs. For these reasons, most blind programmers continue to concentrate on text-based applications. Of 130 blind programmers in the AFB's database, only a dozen work in Windows application development (a 2004 private communication by Christa Earl to U. Obianyo-Agu; see "Notes").

It was the original goal of the project to create a platform-independent RAD (Rapid Application Development) programming language that was suitable for use by blind programmers; no such language exists currently (electronic mail from C. Chong to R. M. Siegfried, see Notes). However, the feedback from members of the **Blind Programming** mailserv list was that they did not need their own programming language; what they needed were tools that would help them work in languages such as Visual Basic. Given the need for such tools, the goal of the project was changed to produce the first of what is hoped to be a series of programming tools to help the blind.

Visual Basic was introduced by Microsoft in 1991 (PC Magazine 2004) and has become a popular application because it allows users to create forms by "pointing and clicking." This popularity has been enhanced by a large number of third party software components available for it. The current version is part of Microsoft's Visual Studio .NET and shares a common runtime library (PC Magazine 2004). Additionally, Visual Basic uses a text file to store information about the forms that an application uses, allowing programmers to make changes in the form without the point and click approach. Forms are stored in text format with the form's properties and its member objects' properties listed together with their values. While this allows one to change these values fairly easily, it is difficult to design a form by creating such a text file. For example, the size of a form is set by specifying the form's height and width in *twips* (twentieths of a point). Similarly, setting the position of a form on the screen is done by specifying the position of the left and top edges in twips.

This presents the blind with the task of providing a large amount of detailed information about the form's layout when creating Visual Basic forms. An example of a simple Visual Basic form appears in Figure 1; the file for this form appears in Appendix A. While it is easy to modify form properties using this file, it is difficult even for a sighted person to create a file like this without using trial and error to determine if the form is correctly constructed. This challenge is even greater for the blind because they cannot see the form. The main goal in a scripting language is to simplify the process of designing a form while allowing users to specify property values that differ from the default.

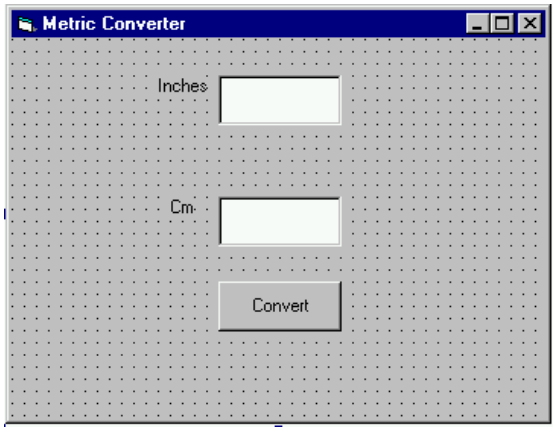

Figure 1: A form for a simple inches-to-centimeter converter

2. THE SCRIPTING LANGUAGE

Using The Scripting Language To Create A Form

The full grammar of the proposed language was specified in 2001 (Siegfried 2002) and the prototype compiler was developed during the summer of 2003. After prototype development was completed, the compiler and the manual were made available to the blind programming community for their feedback. So far, the comments have been positive. The full revised grammar for the language in BNF is specified in Appendix B.

The compiler is a console application under the name **molly.exe**. The files containing the form scripts use the extension **.fms** (for **f**or**m** **s**cript). After creating the form script using any text editor (such as Notepad), the form script can be compiled a command prompt window using the command

molly *FileName***.fms**

where *FileName***.fms** is the name of the form script file. If the file's name is test.fms the command would be:

molly test.fms

This will produce a standard form file **test.frm**, which can be included in a Visual Basic project. The script for the form shown is Figure 1 is:

```
Form InToCm
Location = Top Left
Caption = "Metric Converter"
Sections = Columns
   Section

      TextBox txtInches
      Height = 2 ' expressed in lines
      Width = Medium
      Label = "Inches"
      END

      TextBox txtCm
      Height = 2
      Width = Medium
      Label = "Cm"
      END

      CommandButton cmdConvert
      Caption = "Convert"
      END
   END ' Section
END ' Form
```

The language uses the Basic style of comments where a comment begins after the apostrophe and continues until the end of the line.

The Scripting Language Syntax

The basic layout of a form script is:

FORM *FormName* ↵
Location = *VerticalAttrib HorizontalAttrib*↵
Caption = *"Caption on Title Bar of Form"* ↵
Organization = *{* **Rows** *or* **Columns** *}*↵

SECTION
SectionAttributes
END ' *Comments appear after an apostrophe until the end of the line*
…
END

Where ↵ indicates a carriage return.

The screen is divided into three rows and three columns: the three rows are top, middle and bottom and the three columns are left, center and right. This allows the user to place the form in different areas of the screen without having to measures twips or use trial and error.

Most forms are organized in rows or columns, but usually not both within the same form. For this reason, the user specifies whether the form's organization is in rows or columns. After the organization is specified, each section (either or a row or column, depending on which one the programmer has chosen), is declared with one or more object in the section, which are automatically laid out sequentially within the section. Their exact placement depends on the particular object and its own space requirements. The programmer can specify as few or as many objects as desired in any given section, as long as they all fit within a window. Similarly, the only limit on the number of sections is their ability to fit within the window.

The initial prototype allows the programmer to use any combination of five different object types: command buttons, text boxes, combo boxes, frames, and check boxes. While Visual Basic has several other object types, it seemed prudent to implement these five objects before adding other object types to the scripting language.

Specifying Objects

Each of the five objects that can be specified has its own syntax because the key properties differ from one object to another. The general syntax for an object is:

ObjectType *ObjectName*
Properties
END

where *ObjectType* is **CommandButton**, **TextBox**, **ComboBox**, **Frame** or **CheckBox**. ObjectName is the name that the object will have within the Visual Basic form. It must be unique within the form and should follow the standard name conventions for Visual Basic objects.

Command Buttons: The syntax for a command button declaration is:

CommandButton *cmdButtonName*↵
Caption = "*CommandButtonCaption*" ↵
END

Command buttons have only one property that must be specified, the caption which is a character string enclosed in quotation marks; this text will appear on the button itself. The sample shown below begins with **cmd** to keep within the naming conventions of Visual Basic.

An example of a command button declaration appears below:

CommandButton cmdConvert
Caption = "Convert to Metric"
END

All command buttons are the default size and are to be centered within the row or column in which it is situated on the form.

Combo Boxes: The syntax for a combo box declaration is:

ComboBox *cboComboBoxName*
Width = *{* **Small** *or* **Medium** *or* **Large** *}*
END

The only property specified in the form script is width, which is **small**, **medium** or **large**, which translate to widths of 1215, 1815 and 2415 twips respectively. The height of the combo box is preset and the box will be centered in either the column or row, depending on the organization. Combo box names should begin with **cbo** in keeping with Visual Basic naming conventions. A combo box declaration might resemble the following:

ComboBox cboSample
Width = Small
END

Text Boxes: The syntax for a text box declaration is:

TextBox *txtTextboxName*
Height = *Number of lines (1, 2, …)*
Width = *{* **Small** *or* **Medium** *or* **Large***}*
Label = "*Text Box Label*"
END

The Text Box declaration requires three properties to be specified: the height of the text box, the width and the label that will appear to the left of the text box. What the scripting language treats as one object is really two separate objects on a Visual Basic form: the text box itself and the label that will appear to its left. The Visual Basic naming convention places the prefix **txt** at the beginning of a text box name. The height is given as the number of lines of text that can appear within the box on the form. The programmer specifies an integer in the range of $1 \leq n \leq 5$; any value outside this range will produce an error. Changes in the text box's height will

not influence the height of its label; however, it will reposition the label so that it remains vertically centered in relation to the text box. The width statement allows the programmer to choose either a small, medium or large width for the text box, as it does for the combo box declaration. The label will shift horizontally depending on the width, but its own width will remain unchanged.

A sample text box declaration appears below:

```
TextBox        txtName
Height = 1
Width = Medium
Label = "Enter Last Name"
END
```

Check Boxes: The syntax for a check box declaration is:

```
CheckBox chkCheckBoxName
Caption = "CheckBox Caption"
Height         = Number of Lines
Width  = { Small or Medium or
       Large }
END
```

In addition to its name, the check box has 3 properties that the programmer specifies: the caption, the height and the width, all of which have been discussed above. The naming convention of Visual Basic requires us to begin with the prefix **chk**. The caption appears next to the actual check box on the form. A check box might look like the following:

```
CheckBox       chkSample
Caption = "CheckBox Sample"
Height = 3
Width =  Medium
END
```

Frames: The syntax of a frame declaration is:

```
Frame fraFrameName
Caption = "Frame Caption"
OptionButtonDeclarations
END
```

Frame declarations are different from other object declarations in that they contain one or more option button declarations themselves. The only property that is set for the frame itself is the caption that appears on its border. Additionally, one or more option button declarations are included; their syntax is:

```
OptionButton optOptionButtonName
Caption = "Option Button Caption"
Visibility = { True or False }
END
```

In addition to specifying the caption for the option button, the programmer specifies whether the button is visible. Although other objects can appear inside a frame in a Visual Basic form, at this time the compiler only allows option buttons. The naming convention of Visual Basic requires that frame names begin with the prefix **fra** and option button names begin with the prefix **opt**. A typical frame might contain:

```
Frame fraQuantity
Caption =       "# of Tickets"
OptionButton optNone
   Caption = ""
   Visibility = False
   END

   OptionButton optOne
      Caption = "1"
      Visibility = True
      END

   OptionButton optTwo
      Caption = "2"
      Visibility = True
   END
END
```

Examples: Appearing below is an example of a complete form script:

```
Form frmPayTV
Location = Bottom Center
Caption = "Pay TV Movies"
Organization = Rows
 Section
  ComboBox       cboChannel
   Width = Medium
  END

  CheckBox       chkControl
   Caption = "Parental Control"
   Height = 1
   Width = Small
  END
 END ' Section

 Section
  TextBox        txtPrice
   Height = 2
   Width = Medium
   Label = "Channel Price"
  END

  CommandButton          cmdOrder
   Caption = "Order Now!"
  END
 END ' Section
END ' Script
```

This produces the form shown in Figure 2. The compiler uses a standard spacing of 480 twips to allow room for a frame on any given row, whether or not there is a frame present. This will be addressed after additional feedback from the blind programming community.

If the form script is changed to include a frame with three option buttons, the resulting form is shown in figure 3. The extra space is useful for separating the frame rom adjacent objects on the form. The full script appears in Appendix C.

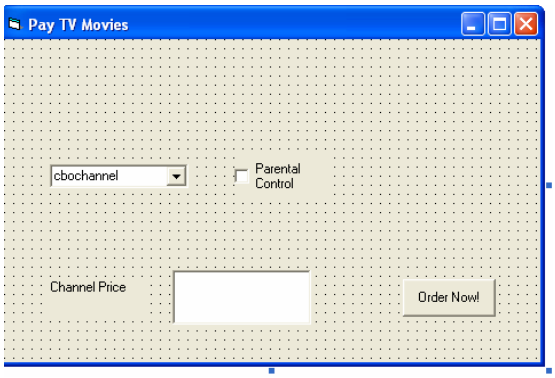

Figure 2: The form created from the above script

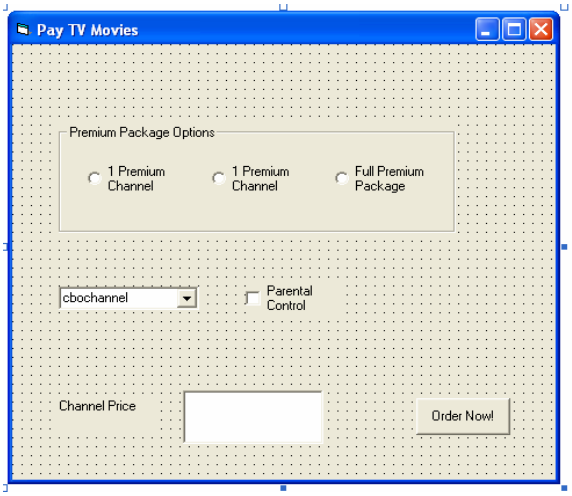

Figure 3 – The form with a frame added to it.

3 DISCUSSION

After posting the prototype compiler and its manual on Adelphi University's web site (http://www.adelphi.edu/~siegfrir/molly) and notifying members of the Blind Programming mail list of its availability, comments were received about the scripting languages; they indicate that the blind programmers who have read the specifications and the sample scripts believe that it has the potential to help them create Visual Basic forms without the aid of a sighted person. At the same time, there were questions about what it might contain in the future. Since this was only an initial prototype, there was never any doubt that additional features needed to be added to the language.

Adding Other Objects to the Language

The five object classes enumerated above were the only ones included in the initial prototype of the scripting language because it was assumed, perhaps erroneously, that these were the most commonly used and the ones whose properties were the easiest to anticipate. It was always expected that other classes of Visual Basic objects would be added to the scripting language later after there was enough feedback to decide if the language as specified met the general needs of the blind programming community. So far the response has been positive, but there is an initial indication that a few changes may improve the language's utility. These changes include:

- *Creating defaults so certain properties need not be specified.* The properties that are specified in each object declaration are required; if they are not included, it will result in a syntax error when the form script is compiled. But there are cases when a programmer may wish to omit a caption, use a standard frame size, such as a height of one line of text and a medium width, or whatever width is sufficient to hold the text.

- *Include some additional properties that may be specified in the scripting language.* It may be advantageous to include other properties in the language, such as initial values for combo boxes. While the programmer can go into the form file and make whatever changes are desired, it defeats part of the scripting language's purpose if programmers have to make common modifications constantly.

- *Allow the scripting language to include statements written in the standard format.* It is difficult to anticipate all the object properties that a user may choose. One unexpected question about the scripting language was whether it could allow the programmer to choose foreground and background colors. The easiest way to deal with this is to allow programmers to set any property for any object in a standard fashion. Given that Visual Basic .NET uses a different format to represent forms than earlier versions of Visual Basic, it may be useful to allow users to specify any object property in a format that is compatible with executable assignment statements in Visual Basic rather than in the format of form files.

- *Include all of the remaining objects.* Given that many commonly used objects, such as scroll bars, list boxes, directory and file list boxes, were not included in the prototypical scripting language, these would have to be added to the language if it were expected to be useful to professional programmers who were blind.

Form Files in Visual Basic .NET
The current version of Microsoft Visual Studio, Visual Studio .NET uses a standard text file format to represent forms in all of the language products that comprise Visual Studio .NET. While this facilitates the use of programs that are developed using more than one programming language, it makes the compiler out of date for those working with Visual Basic .NET. Fortunately, Visual Studio .NET provides a tool for converting Visual Basic forms to the new format; however this complicates the job of creating a single scripting language compiler that is capable of creating forms that are compatible with all formats. Since only a small portion of the compiler is involved in generating the form file, we expect to have a version of the compiler that generates .NET-compatible forms within a few months.

Further testing
Everyone who has worked on the project is sighted. As sensitive as the development team could be to the needs of the blind and visually impaired, it is impossible to anticipate entirely what the blind would consider more suitable for their needs. For this reason, we plan to begin testing within the next few months.

4 CONCLUSIONS

The prototype compiler simplifies the task of creating Visual Basic forms for blind programmers, for whom the "point and click" design method is not possible. Although testing has not been completed, preliminary comments suggest that additional features will make it a useful tool for the blind. These features allowing the scripts to specify object not currently included in the language and default values for certain properties. Additionally, a version of the compiler that can create form files compatible with Visual Studio .NET is needed as well as the current version, which creates form files compatible with version 6.

The compiler and manual are currently available online at http://www.adelphi.edu/~siegfrir/molly.

5. NOTES

Christa Earl is blind and works as a web site developer for the American Federation of the Blind in New York.

Curtis Chong is President of the National Federation of the Blind in Computer Science, Des Moines, IA, USA.

6. REFERENCES

Ceruzzi, Paul E., 2000, A History of Modern Computing, The MIT Press, Cambridge.

Kirchner, C., & E. Schneidler, 1997,"Prevalence and Employment of People in the United States Who Are Blind or Visually Impaired", Journal of Visual Impairment and Blindness 91(5), Sept/Oct, p.508-511.

Palfreman, Jon, and Doron Swade, 1991, The Dream Machine: Exploring the Computer Age, BBC Books, London.

PC Magazine, 2004. Retrieved June 17 from http://www.pcmag.com/article2/0,1759,15012,00.asp

Siegfried, Robert M., 2002, "A Scripting Language To Help The Blind To Program Visually", ACM SIGPLAN Notices 32(2), February, p. 53-56.

Wallace, James, and Jim Erickson, 1993, Hard Drive: Bill Gates and the Making of the Microsoft Empire, HarperBusiness, New York.

Appendix A - The Visual Basic file for the form in Figure 1.

```
Begin VB.Form frmInToCm
  Caption        =   "Metric Converter"
  ClientHeight   =   3750
  ClientLeft     =   60
  Clients              =   345
  ClientWidth    =   5265
  LinkTopic      =   "Form1"
  ScaleHeight    =   3750
  ScaleWidth     =   5265
  StartUpPosition =   3  'Windows Default
  Begin VB.CommandButton cmdConvert
    Caption        =   "Convert"
    Height         =   495
    Left           =   2040
    TabIndex       =   4
    Top            =   2400
    Width          =   1215
  End
  Begin VB.TextBox txtCm
    Height         =   495
    Left           =   2040
    TabIndex       =   2
    Top            =   1560
    Width          =   1215
  End
  Begin VB.TextBox txtInches
    Height         =   495
    Left           =   2040
    TabIndex       =   0
    Top            =   360
    Width          =   1215
  End
  Begin VB.Label lblCm
    AutoSize       =   -1  'True
    Caption        =   "Cm"
    Height         =   195
    Left           =   1560
    TabIndex       =   3
    Top            =   1560
    Width          =   225
  End
  Begin VB.Label lblInches
    AutoSize       =   -1  'True
    Caption        =   "Inches"
    Height         =   195
    Left           =   1440
    TabIndex       =   1
    Top            =   360
    Width          =   480
  End
```

Appendix B - A BNF grammar for the scripting language

Form ::= *Header FormAttributes OrgAttributes SectionDeclarations* **end**
Header ::= **form id** *Returns*
FormAttributes ::= *LocationAttribute CaptionAttribute*
LocationAttribute ::= **location =** *VerticalAttribute HorizontalAttribute Returns*
VerticalAttribute ::= **top** | **middle** | **bottom**
HorizontalAttribute ::= **left** | **center** | **right**
CaptionAttribute ::= **caption** = String *Returns*
OrgAttributes ::= **organization =** *SectionOrg*
SectionOrg ::= **rows** | **columns**
SectionDeclarations ::= *SectionDeclarations SectionDeclaration* | *SectionDeclaration*
SectionDeclaration ::= **section** *Returns ObjectDeclarations* **end** *Returns*
ObjectDeclarations ::= *ObjectDeclarations ObjectDeclaration* | *ObjectDeclaration*
ObjectDeclaration ::= *CommandButtonDeclaration* | *TextBoxDeclaration* | *ComboBoxDeclaration* | *FrameDeclaration* | *CheckBoxDeclaration*
CommandButtonDeclaration ::= **commandbutton id** *Returns CaptionAttribute Returns* **end** *Returns*
TextBoxDeclaration ::= **textbox id** *Returns SizeAttributes LabelAttribute* **end** *Returns*
SizeAttributes ::= *HeightAttribute WidthAttribute*
HeightAttribute ::= **height =** *Number Returns*
WidthAttribute := **width =** *Size Returns*
Size ::= **small** | **medium** | **large**
LabelAttribute ::= **label =** *String Returns*
ComboBoxDeclaration ::= **combobox id** *Returns WidthAttribute Returns* **end** *Returns*
FrameDeclaration ::= **frame id** *Returns CaptionAttributes OptionDeclarations* **end** *Returns*
OptionDeclarations ::= **optionbutton id** *Returns CaptionAttribute VisibleAttribute* **end** *Returns*
VisibleAttribute ::= **visible =** *Boolean Returns*
Boolean ::= **true** | **false**
CheckBoxDeclaration ::= **checkbox id** *Returns CaptionAttribute SizeAttributes*
Returns ::= *Returns* ↵ | ↵
String ::= **"** *AlphaNumeric** **"**
AlphaNumeric ::= Letter | Digit
Number ::= *Digit Digit* *
Letter ::= A | B | … | Y | Z | a | b | … | y | z
Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

↵ indicates the newline character

Appendix C – the full script for the form in Figure 3.

```
Form frmPayTV
Location = Bottom Center
Caption = "Pay TV Movies"
Organization = Rows
  SECTION
     FRAME  fraPremium
     CAPTION = "Premium Package Options"
      OPTIONBUTTON optNone
      CAPTION = "1 Premium Channel"
      VISIBLE = TRUE
      END ' option button
      OPTIONBUTTON optOne
      CAPTION = "1 Premium Channel"
      VISIBLE = TRUE
      END ' option button
      OPTIONBUTTON optAll
      CAPTION = "Full Premium Package"
      VISIBLE = TRUE
      END ' option button
     END ' frame
  END  ' section
  SECTION
     ComboBox       cboChannel
     Width = Medium
     END ' combobox
     CheckBox       chkConrol
     Caption = "Parental Control"
     Height = 1
     Width = Small
     END ' combobox
  END ' section
  SECTION
     TextBox txtPrice
     Height = 2
     Width = Medium
     LABEL = "Channel Price"
     END ' text box
     CommandButton cmdOrder
     CAPTION = "Order Now!"
     END ' command button
  END ' section
END ' form
```