Jl. of Educational Multimedia and Hypermedia (2002) **11**(3), 221-235

An Introduction to Custom WebBrowsers for the Qualitative Study of Hypertext Navigation

PATRICIA M. BOECHLER AND MICHAEL R.W. DAWSON
University of Alberta, Canada
patricia.boechler@ualberta.ca

KELVIN R. BOECHLER
Saskatchewan Institute of Applied Science and Technology
Canada

For the benefit of researchers new to custom programming, this article presents a discussion of the advantages of custom applications and an example of such a program with a description and explanation of some of the programming code used to create it. *Visual Basic 6.0* is an Integrated Development Environment (IDE) that allows researchers to custom design interface forms using standard window controls and to code routines using the *Visual Basic* programming language. *Visual Basic 6.0* (Professional Edition) also includes specialized controls such as the WebBrowser Control that enables researchers to add hypertext navigation to an application with a highly adaptable browser interface. This collection of functions provides a particularly valuable tool for investigating hypertext navigation and user interactions with document features.

The term “hypertext” refers to any electronic document or document collection of interconnected units of information. A prominent example of a large hypertext system is the World Wide Web (WWW or Web). On a smaller scale, customized, constrained systems of hypertext documents (intranet systems) have become commonplace in work and educational environments.

Unfortunately, the cognitive processes involved in navigating through even a single hypertext document are not understood clearly. This is largely because the task of navigating in hypertext involves the integration of many complex cognitive processes within a very multifaceted environment. The multitudes of variables that a navigation task presents demands methodologies that control the interface users see and provides flexibility for collecting diverse types of data. This article outlines a custom application, programmed using *Visual Basic 6.0* that provides both of these benefits.

Controlling the Interface

The *Visual Basic* ActiveX WebBrowser Control included in *Visual Basic 6.0* creates a highly adaptable web browser that allows for the management of options available to users of hypertext documents. This is important for two reasons. First, to access information about the cognitions involved in hypertext use, a pared-down, sparse version of browsers and hypermedia pages may be desirable. Custom programming allows researchers to introduce new elements to the browser controls as well as eliminate unwanted controls resulting in interface elements that are specifically applicable to the question of interest. For example, for some research questions, the elimination of options such as the back, forward, or print buttons from the browser menu may be favorable.

Second, a custom browser ensures a novel environment for every user regardless of their prior experiences with standard browsers. Pretesting users on previous computer use is helpful but these measures usually rely on general information such as how often a computer is used within a certain timeframe or the number of activities a user engages in on a regular basis (e.g., e-mail, word processing). Questionnaires can result in ambiguous responses depending on the questions asked and a user's interpretations of the questions. Ensuring that every user begins the research task with an equal level of familiarity with the browser eliminates one extraneous variable.

Collecting Appropriate Data

Often, hypertext researchers are interested in tracking very quantitative navigation responses such as time spent on pages, number of pages accessed, and order of pages. A custom program can provide procedures for data collection that are strictly controlled by a researcher, eliminating any impact

of features inherent in other standard tracking mechanisms such as browser history lists or server logs.

For example, browser history lists make use of a variety of procedures for recording page access. To optimize the history list, some methods erase prior revisits and only retain the most recent instance of access (for further description of history lists see Tauscher & Greenberg, 1997). By creating a custom application, a local log file of users' movements can be constructed to record every single move with no elimination of repeated page accesses that can happen with the typical history lists of standard browsers.

For some types of measures, collecting data through server logs presents obstacles as well. The procedures used by the server to cache items may skew data by affecting the way pages are delivered to the user. For example, as numerous users access a single page numerous times, the server retains the page in memory and will load that page quickly relative to a page that is not held in memory. This results in navigation experiences that may be different from user to user. Also, because server use involves multiple users on a shared network, system traffic could interfere with timed results as page delivery could be delayed by network congestion. By having tracking procedures local to each individual work station, that is, creating local logs instead of server logs, the above-mentioned problems can be avoided.

The program outlined in this article also allows for seamless transitions between the browser that displays web pages and additional cognitive tasks that provide other types of information about a user's navigation experience. For example, in the study outlined, (Boechler & Dawson, 2002a; Boechler & Dawson, 2002b) users are required to complete an information search task of 10 questions using a customized browser. During this task, navigation data is collected as well as a user's answers to the ten questions. Users use the mouse to select sections of text as their answer to each question. This method avoids data that is difficult to process due to users' typing mistakes or inconsistent use of terms. When this task is complete, the program switches from the browser to a series of psychological tasks about the document—a free recall task, a recognition task and a distance-like ratings task recording all responses in a flexible format (text files) for subsequent data analysis.

Advantages of *Visual Basic 6.0*

It is true that other programming formats include an adaptable browser function, so why *Visual Basic 6.0*? For researchers new to custom applications, there are several pragmatic reasons for using *Visual Basic*. First, it is

easily accessible. Versions of *Visual Basic* are included in popular office suite software packages. Second, *Visual Basic* is well–supported through reference books and extensive online help. Third, *Visual Basic* is a relatively simple programming environment, compared to many other programming languages.

Educational and psychological researchers are faced with an often-confusing plethora of technological possibilities for collecting relevant hypertext data. This article is meant to provide a starting point for researchers who may wish to pursue the use of custom applications. It provides an example of a computer program that uses a relatively simple programming environment to gather basic navigation data as well as collect data on other behaviors that may be of interest to the hypertext researcher.

First, the development of the interface of the document is described. Second, the coded routines that accompany the interface are detailed.

INTERFACE DEVELOPMENT

The hypertext interface is created in two parts: (a) the actual hypertext pages with the specified content and links are constructed and (b) the *Visual Basic* application forms, including the WebBrowser Control that displays the web pages, are programmed.

Developing the *Visual Basic* Application

Visual Basic 6.0 is an Integrated Development Environment (IDE) that allows a researcher to design, run, test, and debug Windows applications using the Visual Basic language. Development in the *Visual Basic 6.0* environment involves placing standard components onto a visual container called a form, thereby constructing the user interface. Coded routines are written to respond to a user's interactions with the form's components. Each control has a certain set of events that will generate responses (eg., mouse click, resized, document loaded). Several forms, which appear as separate windows, were constructed for this application, including the form that contains the specialized component that displays web pages, the WebBrowser control.

The Development of Forms

Individual forms for the interface are constructed to accomplish certain tasks. Tasks are subdivided into user actions that are represented as various controls in strategic positions on the form. Each form in the application is maintained in a file designated by the form name plus the extension .frm. Form files contain the graphic interface of the form and also text files that contain code to describe the controls on the form and the events that correspond to them. During the development phase, switching between the graphic interface and the code window of the form is easily accomplished with only one icon selection.

There are several types of standard window controls that can be included on a *Visual Basic* form. For example, Textboxes are used to display and enter text. Command buttons represent actions that will occur when a user activates that control. Option buttons are presented in groups and are used when a user is required to make one choice out of several options. The CheckBox control allows a user to make several choices from a list of options. In Figure 1, the Administrative Login form shows an example of a form with labeled textboxes (the spaces provided for user and password submissions), option buttons (the buttons used to select the appropriate condition) and a command button (the "submit" button). Such controls are added by selecting and placing icons from the General Toolbox menu onto the form. Numerous properties can be selected for each control (e.g., for the textbox control, properties for the number of text lines, the inclusion of scroll bars, and the maximum length of the entered text can be chosen). Properties can either be set from the Properties window or written into the code. As the project is developed, the Project Explorer provides a list of the components (e.g., forms) that have been added to the project.

CODE THAT ACCOMPANIES THE INTERFACE

Controls can be activated by several types of events triggered by user interactions in the form of either mouse use (e.g., clicking, press and release of the mouse button, or movement of the mouse) or keyboard use (e.g., key press, changes to text boxes). Other events are triggered by changes in control state (e.g., the WebBrowser's documentComplete is triggered when the control has successfully loaded a page). Whenever a control on a form is activated, the program will initiate the event routines that accompany that control. For example, on the Login form (Figure 1), once selections are made

between the option buttons in the dialog boxes, clicking on the "submit" button initiates the subroutine coded for the "on click" event as outlined in Listing # 1. Note the initial section of code for each form declares the variables local to that form using the *Dim* (dimension) statement. Local variables that are intended to be available only to a specific subroutine can also be declared within individual subroutines.

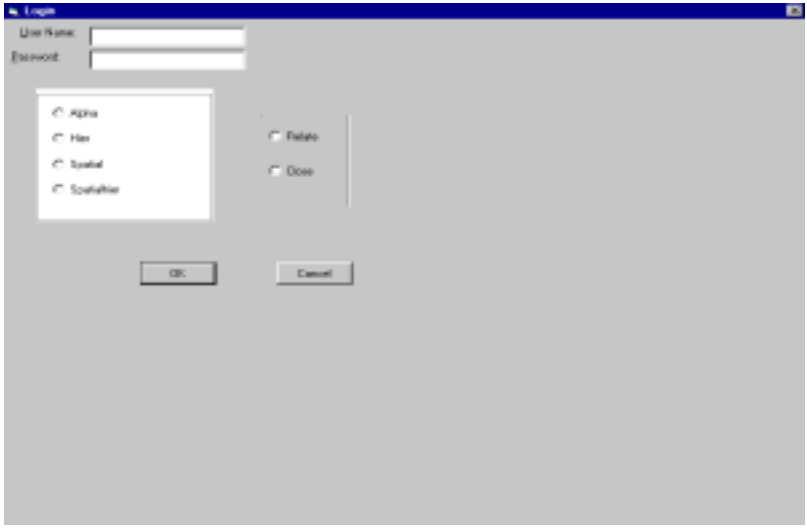

Figure 1. Administrative login form

An initial administrative login form, such as the one previously described, can be used to specify a particular condition or version of stimuli for each individual subject. Later in the application, flow control statements (e.g., "If-Then," "Select Case") can access the selections made in the initial Login form and use them to execute only the appropriate statements for the specified stimuli. In addition to the administrative login form, this application includes a user login form that collects identification data particular to a given subject (see Figure 2).

LISTING #1—ADMINISTRATION LOGIN FORM CODE

```
Option Explicit
    Public LoginSucceeded As Boolean
    Dim QuestionSelected As Boolean
    Dim ConditionSelected As Boolean

    Private Sub cmdOK_Click()
    '*********************************************************************
    'Event: User clicks OK button
    'Response:
    '    If the password is correct
    '            Record login success
    '            Check if one of the navigation aid conditions is selected
    '            Check if one of question wording options is selected
    '            If both question wording and navigation condition have been
                selected
    '               Close this form
    '               Move to the test subjects login (LoginForm.Show)
    '            Otherwise
    '               Indicate required information is missing
    ' Otherwise
    '            Indicate the password is incorrect
    '            Set up the form for the user to re-enter password
    '            Hide Administrative Login Form
    '            Show test subject login form (LoginForm.Show)
    '*********************************************************************
    If (txtPassword.Text = Psswrd) Then
            LoginSucceeded = True
            ConditionSelected = ((Option1(0).Value) Or (Option1(1).Value) _
                    Or (Option1(1).Value) Or (Option1(2).Value) Or
                    (Option1(3).Value))
            QuestionSelected = ((Option2(0).Value) Or (Option2(1).Value))
            If (QuestionSelected) And (ConditionSelected) Then
                Me.Hide
                LoginForm.Show
            Else
                MsgBox "Something's Missing!"
            End If
    Else
            MsgBox "Invalid Password, try again!", "Login"
            txtPassword.SetFocus
            SendKeys "{Home}+{End}"
    End If
    End Sub
```

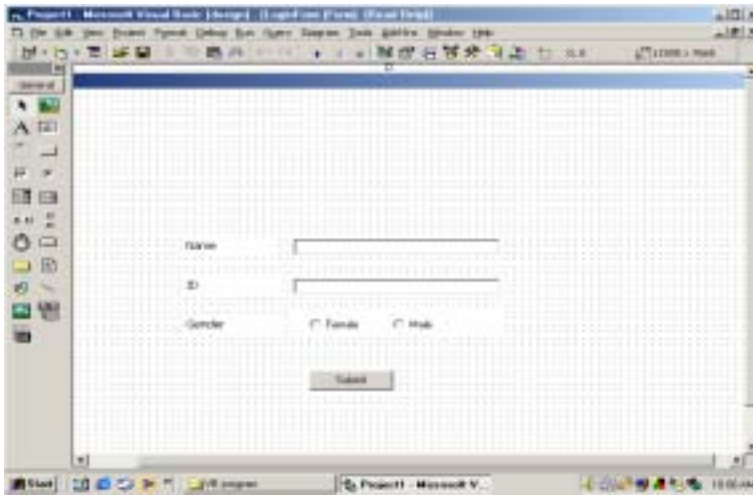

Figure 2. User login form

Displaying the Web pages

When the Browser Form is loaded, the "select case" structure is used to direct the program to present the appropriate instruction page and the navigation aid to be used in that specific test session (Listing # 2, Subroutine #1 and #2 from the Browser Form set-up routines). The four possible conditions in this study were named Alpha, Hier, Spatial, Spatialhier.

LISTING # 2—SUBROUTINE# 1—BROWSER FORM CODE (SET-UP)

```
Dim StartTime As String
Dim EndTime As String
Dim PageTime As String
Dim LastPage As String
Dim Answer As String
Dim Question As Byte
Dim Page As String
Dim InstructPage As String
Dim Done As String

Private Sub Form_Load()
'*********************************************************************
'Event: Form Loads
'Response:
'    Initialize start and end times for tracking user
```

```
‘    Open navigation log files
‘    Select instruction page based on admin options set in AdminLogin
‘    Navigate to the instruction page
‘*********************************************************************
Let EndTime = Timer
Let StartTime = Timer
Open (App.Path & AnswerFile) For Append As #3
Open (App.Path & LogFile) For Append As #2
Let Frame1.Visible = False
Let Submit.Visible = False
Let LastPage = App.Path & InstructFile
Select Case Condition
   Case “Hier”, “Alpha”
            InstructPage = “\instructions2.htm”
   Case “Spatial”, “Spatialhier”
            InstructPage = “\instructions1.htm”
 Case Else
            InstructPage = “\instructions1.htm”
End Select
WebBrowser1.Navigate App.Path & InstructPage
End Sub
```

LISTING # 2—SUBROUTINE# 2—BROWSER FORM CODE (SET-UP)

```
Private Sub Continue_Click()
‘*********************************************************************
‘Event: Continue button clicked
‘Response:
‘    Set up form for questions
’    Select navigation aid page based on admin options set in AdminLogin
‘    Navigate to the navigation aid page
‘*********************************************************************
Question = 1
Frame1.Visible = True
Continue.Visible = False
Change_Question (Question)
Select Case Condition
   Case “Hier”
            Page = “\hierlist5.htm”
   Case “Alpha”
            Page = “\alpha.htm”
   Case “Spatial”
            Page = “\spatial.htm”
   Case “Spatialhier”
            Page = “\SpatialHier.htm”
   Case Else
            Page = “\alpha.htm”
End Select
WebBrowser1.Navigate App.Path & Page
End Sub
```

Loading the Browser Form also initializes the timer, opens the appropriate data files, and presents the desired controls on the form (e.g., "submit" command button). The final line of code in the second subroutine (Listing #2, subroutine #2) of the Browser Form instructs the WebBrowser Control to navigate to the designated page, which appears in the window at run time. Once the WebBrowser Control is functioning, the links embedded in the individual web pages allow a user to revisit the navigation aid and instructions page. Each successful navigation triggers the WebBrowser's "Document Complete" event, which causes the subroutine WebBrowser1_DocumentComplete to execute, calculating, and recording the time a user has spent on the previous page (Listing #3, Subroutine #5, #6).

LISTING # 3—SUBROUTINE #5, #6—BROWSER FORM CODE (OPERATIONAL)

```
Private Sub WebBrowser1_DocumentComplete(ByVal pDisp As
        Object, URL As Variant)
'*****************************************************************
'Event: User navigation to new document page
'Response:
'    Log time taken on last page
'    Prepare for next log entry by recording the page navigated to
'       as the last page visited
'*****************************************************************
LogTime
Let LastPage = URL
End Sub


Private Sub LogTime()
'*****************************************************************
'Event: Called directly by coded routines
'Response:
'    Get the time spent on the last page and record it in the log
'    Reset the start time for the next page
'*****************************************************************
Let EndTime = Timer
Let PageTime = EndTime - StartTime
Write #2, SubjectInfo.Name, LastPage, PageTime
Let StartTime = Timer
End Sub
```

Developing Data Files

For the program to open data files and store data within them, certain variables must be accessible to all Forms and Modules throughout the run time of the application. These variables must be declared as *Public* (rather than *Dim*) and as Form variables or within Modules. An example of public variables declared within a module appears in Listing #4. This module also declares the set value for the password for the initial administrative login.

LISTING #4—MODULE #1 FOR DECLARATION OF PUBLIC VARIABLES

```
'*********************************************************************
'Module: Module1
'File: Module1.bas
'Purpose: To store and consolidate variables and constants required by all
        forms
'*********************************************************************
Type SubjectData
        Name As String * 30
        ID As Long
        Gender As String
End Type
Public SubjectInfo As SubjectData
Public Const SubjectFile = "\Subject.txt"
Public Const LogFile = "\LogFile.txt"
Public Const AnswerFile = "\Answers.txt"
Public Const Recallfile = "\Recall.txt"
Public Const RecogFile = "\Recog.txt"
Public Const RatingsFile = "\Ratings.txt"
Public Const InstructFile = "\Instructions.htm"
Public Const Psswrd = "space"
Public RatingsQuestion As String
Public Condition As String
```

In the study described, users' responses to four tasks were collected and stored in text files that were automatically constructed when the user interacted with specific forms in the program. For three of the tasks (the free recall, recognition, and ratings tasks), this merely involved transferring the typed responses or option button and checkbox responses of the user to text files. The fourth task, the initial information search task, required the storage of selections of text, which necessitated a slightly different approach. In order to check the accuracy of a user's search efforts, users were instructed to highlight the answer with the mouse (e.g., phrase or sentence) for each question

from the page on which they found it and then submit the answer by clicking on a "submit" command button. The code that responds to the "Click" event takes advantage of the fact that the WebBrowser is an OLE (Object Linking and Embedding) container for the active HTML page. An OLE control allows the programmer to insert objects from other applications into the *Visual Basic* form, so the functions of that application are available to the VB program. In this case, the OLE container accesses the *Visual Basic* document through an interface that initiates the correct implementation of a common command. One of the commonly supported commands is to copy the currently highlighted or selected text to the clipboard. Using the execCommand method of the active document's interface with "Copy" as the indicated command, the highlighted or selected data is copied to the clipboard. The globally available (common to and useable by all) clipboard was then accessed by the research program, which retrieved the data to a local variable and wrote the information to file (Listing # 3, Subroutine #3, #4). Constraining a user to copying and pasting information to the clipboard avoids common problems like typing mistakes or user interpretation of data, which would make subsequent data verification or validation difficult.

LISTING # 3—SUBROUTINE #3—BROWSER FORM CODE (OPERATIONAL)

```
Private Sub Found_Click()
'*********************************************************************
'Event: Found answer button is clicked
'Response:
'    If attempting to select answer from the navigation aid
'       Indicate this action is in error and continue current question
' Otherwise
'    Log time taken on last page
'    Set up for user to submit the answer
'*********************************************************************
If (("\" & WebBrowser1.LocationName) = Page) Then
   MsgBox ("Answers cannot be found on this page.")
Else
   LogTime
   Found.Visible = False
   Submit.Visible = True
End If
End Sub
```

LISTING # 3—SUBROUTINE #4—BROWSER FORM CODE (OPERATIONAL)

```
Private Sub Submit_Click()
'*****************************************************************
'Event: Submit answer button is clicked
'Response:
'    Empty the clipboard
'    Copy the user highlighted information from the document
'            to the Answer variable by having the WebBrowser's document
             object
'            use its ExecCommand to perform a standard command, in this
             case copy
'    If there is nothing in Answer then
'            Indicate to the user there has been a mistake and exit
'    Otherwise
'            Write the information to the log files
'            If that was the last question then
'                    Log the time for this last event
'                    Move to title recall form
'    Otherwise
'            Change to the next question
'            Set up to allow user to select found answer
'            Reset new page start time
'            Navigate to navigation aid
'*****************************************************************
Clipboard.SetText ("")
WebBrowser1.Document.ExecCommand  ("Copy")
Let Answer = Clipboard.GetText
Let Done = "Done"
If (Answer = "") Then
   MsgBox ("Please highlight the answer before clicking submit")
Else
   Write #3, SubjectInfo.Name, Question, Answer
   Write #2, Done
Question = Question + 1
   If (Question = 11) Then
            LogTime
            Me.Hide
            TitleRecall.Show
 Else
            Change_Question (Question)
            Submit.Visible = False
            Found.Visible = True
            WebBrowser1.Navigate App.Path & Page
End If
End If
StartTime = Timer
End Sub
```

The collection of efficiency measures was accomplished using a log file of a user's movements through the document which records each page accessed including revisitations, the time spent on each page and the page where a user found the information required by the search task (Figure 3 an example of the log file). The log file was constructed by writing successive lines in the log file each representing a record of specific structure (eg., User, Page, Time Spent). Each record is a series of comma delimited pieces of information. The new line character separates individual records. The format of the log files was chosen because it offers a format that is tool independent providing guaranteed access while being of a common format used by data analysis tools that import data from other sources. This allows for easy organization of data beyond the data collection phase.

```
"Subject 1 ","C:\VB Project2\Instructions.htm","0.719999999993888"
"Subject 1 ","C:\VB Project2\instructions1.htm","47.5800000000017"
"Subject 1 ","C:\VB Project2\SpatialHier.htm","19.4700000000012"
"Subject 1 ","C:\VB Project2\Shtypes.htm","25.9100000000035"
"Subject 1 ","C:\VB Project2\SpatialHier.htm","4.63999999999942"
"Subject 1 ","C:\VB Project2\Shclub.htm","18.6100000000006"
"Subject 1 ","C:\VB Project2\SpatialHier.htm","1.44999999999709"
"Subject 1 ","C:\VB Project2\Shsac.htm","23.9499999999971"
"Subject 1 ","C:\VB Project2\SpatialHier.htm","1.68000000000029"
"Subject 1 ","C:\VB Project2\Shconjugat.htm","23.0600000000049"
"Subject 1 ","C:\VB Project2\SpatialHier.htm","1.83999999999651"
"Subject 1 ","C:\VB Project2\ShChytrids.htm","26.1800000000003"
"Subject 1 ","C:\VB Project2\SpatialHier.htm","1.13999999999942"
"Subject 1 ","C:\VB Project2\Shlichen.htm","24.5999999999985"
```

Figure 3. Example of the logfile Data

CONCLUDING REMARKS

The Visual Basic program outlined provides an accessible and effective means of tracking the processes involved in hypertext use. Once the custom browser program is created, altering the content and functions of the individual hypertext pages can modify the task to address additional research questions. To accompany navigation data, relevant tasks such as recall and recognition tasks can be programmed for online data collection in a text format that is easily manipulated for subsequent data analysis.

References

Boechler, P.M., & Dawson, M.R.W. (2002a). Connections between performance, path patterns and mental representations in hypertext navigation. *Journal of Educational Multimedia and Hypermedia, 11*(2), 95-115.

Boechler, P.M., & Dawson, M.R.W. (2002b). *The effects of navigation tool information on hypertext navigation behavior: A configural analysis of page-transition data.* Manuscript in preparation.

Tauscher, L., & Greenberg, S. (1997). How people revisit web pages: Empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies, 47*, 97-137.