JOURNAL OF APPLIED BEHAVIOR ANALYSIS 2007, **40,** 359–384 NUMBER 2 (SUMMER 2007)

A MOBILE COMPUTING SOLUTION FOR COLLECTING FUNCTIONAL ANALYSIS DATA ON A POCKET PC

JAMES JACKSON AND MARK R. DIXON

SOUTHERN ILLINOIS UNIVERSITY

The present paper provides a task analysis for creating a computerized data system using a Pocket PC and Microsoft Visual Basic. With Visual Basic software and any handheld device running the Windows Moble operating system, this task analysis will allow behavior analysts to program and customize their own functional analysis data-collection system. The program will allow the user to select the type of behavior to be recorded, choose between interval and frequency data collection, and summarize data for graphing and analysis. We also provide suggestions for customizing the data-collection system for idiosyncratic research and clinical needs.

DESCRIPTORS: computers, data collection, functional analysis, Microsoft Visual Studio

Data collection is a key characteristic of behavior analysis. Data guide clinical decision making and provide evidence of scientific discovery. Recently there have been a number of commercial software programs that have emerged to make the job of data collection easier for behavior analysts (see Kahng & Iwata, 1998, for a review). As computers become smaller, more commonplace, and less expensive, mobile computer solutions for aiding the behavior analyst continue to grow year after year. Dixon (2003) described a means by which a reader could create his or her own data-collection system using a Pocket PC and Microsoft Embedded Visual Basic. Although this paper provided task analyses for obtaining the necessary software, programming the computer to collect duration and latency data, deploying the software to a Pocket PC, and downloading the data for further analysis, the paper did not provide readers with a flexible program that could be used under typical clinical and applied research conditions.

Functional analyses (Iwata, Dorsey, Slifer, Bauman, & Rchman, 1982/1994) have risen in usage over the past decade such that their inclusion in behavior treatment plans is often required (Erchul & Martens, 2002), presumably because they provide the foundation for effective behavioral interventions (e.g., DeLeon, Arnold, Rodriguez-Catter, & Uy, 2003; Hanley, Iwata, & McCord, 2003). Data are often collected with laptop computers during functional analyses, which can be difficult if the observer is also required to deliver programmed consequences. Furthermore, full-size computers or laptops are often impractical when data are collected in multiple sites, with multiple clients, and with clients who may attempt to destroy equipment during experimental sessions.

Therefore, the purpose of the present paper is to provide readers with a straightforward description of how to create a customized, portable, and computerized data-collection system for a Pocket PC that may be particularly, but not exclusively, relevant to conducting functional analyses. This system can be programmed by persons with general computer knowledge of the Microsoft operating and software systems. The program described in this paper uses Microsoft's 2005 version of Visual Basic programming language. The Visual Basic 2005 language has replaced the Embedded Visual Basic software described by Dixon (2003), thus creating a need for description of the new software's features for interested behavior analysts.

This program will also function correctly with the Windows Vista operating system.

Address correspondence to Mark R. Dixon, Behavior Analysis and Therapy Program, Rehabilitation Institute, Southern Illinois University, Carbondale, Illinois 62901 (e-mail: mdixon@siu.edu).

doi: 10.1901/jaba.2007.46-06

Table 1
Summary of Hardware, Software, and User Skills Necessary to Complete Program

Minimum hardware requirements	Minimum software requirements	Minimum computer skills necessary	Minimum time commitment to complete program	Expected skills acquired upon program completion
Pocket PC handheld computer running Microsoft CE 2003 or higher	Microsoft XP professional or home operating system	General familiarity with the Windows operating system and office program suite (i.e., Word, Excel, etc.)	Mean of 4 hr (range, 2–6 range) for entire program from start to finish	Ability to create and implement a computer-based data-collection system
Desktop or laptop computer with minimum of 512k RAM and Pentium-based processor	Microsoft Visual Studio 2005 or Visual Basic 2005 or higher version	General familiarity with Windows software menu options (i.e., save files, cut and paste text, view program options)	15 min for software transfer to Excel for data analysis	Ability to modify the program to individual data-collection needs
Sound card (optional) is required for programming auditory stimuli			30 min to download interobserver agreement program from *JABA* website and install on desktop	An introductory understanding of the general programming techniques in Visual Basic

HARDWARE, SYSTEM, AND SOFTWARE REQUIREMENTS

Readers interested in constructing this functional analysis data-collection system should refer to Table 1 for a list of hardware and software requirements, along with expected skill level and time commitments. In summary, the user should have (a) a Pocket PC handheld computer that runs Windows Mobile 2003 or higher operating system, (b) a desktop or laptop computer that runs the Microsoft XP professional or home operating system, and (c) purchased and correctly installed Microsoft Visual Studio 2005, Visual Basic 2005, or a higher version of this programming software.

Although the Visual Studio 2005 software will run on any computer that can operate the XP operating system, we recommend that the user have a minimum of 512k RAM, a sound card, and a Pentium-based processor. Faster computers will make programming and testing the system quicker.

The upfront costs for undertaking the programming described in this paper will include the price of a desktop computer if none is currently available. Besides this cost, which has broader use and utility beyond the current project, a Pocket PC may cost between $200 and $500, and the Visual Studio software will cost around $100 for the student/academic version and up to $500 for the professional version. The student/academic version contains all the necessary tools to complete this project and most other projects of interest to behavior analysts (see Dixon & MacLin, 2003, for desktop programming in Visual Studio.NET). We have programmed this system using a $200 Pocket PC from Dell Computers with the student version of the software purchased for $99. We also field tested the program with various other brands of Pocket PCs as well as programmed the software using the professional and enterprise versions of Visual Studio 2005.

OVERVIEW OF TASK ANALYSIS

We will first describe the overall rationale and purpose of the application. We will then illustrate how to start an application in Microsoft Visual Studio 2005 and give an overview of some of the controls that can be used to build applications. We will then describe how to create the needed interfaces and controls for the current project and provide the essential computer code for the application. Finally, we will provide

Table 2
A Summary of Possible Problems and Possible Solutions to Programming for the Pocket PC in Microsoft Visual Basic 2005

Problem	Possible solution
Software fails to deploy on Pocket PC	Check communications cable, check to make sure Pocket PC is firmly in docking cradle, make sure Microsoft ActiveSync is running on desktop computer
Software takes unusually long time to compile onto Pocket PC	Free additional memory space on Pocket PC, upgrade version of Windows CE or Windows Mobile
Pocket PC emulator does not work or generates error	Test program on actual Pocket PC
Data output file cannot be found	Check "My Device" directory on Pocket PC, or use "Find Files Option" on desktop if using emulator
Each subsequent observation overwrites old file	Add a textbox on first form for "Observation Number," declare a "Public" variable on module for observation number, make observation number part of output file name
User wants to add ability to end program before session timer reaches end of its interval	Add command button named "End," copy all of the code for session timer into button's subroutine, exit program with switcher box
Switcher bar not present	Click on Windows Icon in upper left of Pocket PC screen, select settings, select "System" tab, locate and click on switcher bar icon
Clicking "Start" button on first form generates error and terminates program	Make sure input file for subject resides in "Input Files" folder, make sure to use same spelling as input file name for the participant
lblHide fails to disappear at end of observation interval	Check code under tmrObserve, make sure it includes lblHide. Visible = False
lblHide disappears prematurely during observation interval, intervals are written multiple times to output file	Check code under tmrRecord, make sure it includes tmrRecord. Enabled = False
Proper number of recording buttons fails to appear for given client	Make sure number at top of input file corresponds with number of behaviors, check select case statement under frmRecord_Load
More buttons for recording behavior are present than are required	Check each button's "Visible" property and make sure it is set to false
Text of behavior doesn't fit on button	Resize button, abbreviate behavior text
Text of behavior doesn't show up on button	Check select case statement under frmRecord_Load
During interval recording, buttons are not enabled on subsequent intervals	Check code under tmrObserve, make sure each button's "Enabled" property is reset to true
Output file for interval recording session shows behavior occurring more than once in interval	Check code under tmrRecord, make sure behavior occurrence variable are reset to zero
Cannot find the data file to transfer into Excel	Use the "Find Files" option, make sure Excel's file types option "All Files" is selected
Data in Excel all appear in one row	Make sure to import data using the "delimited" and "comma" options in Excel

guidance for installing the program on a Pocket PC and describe how the output files can be imported into a statistical program such as Microsoft Excel for data analysis and graphing. Throughout each area of the task analysis, we will provide the reader with prompts for building the application, rationale for sections of code, and descriptions of areas that might allow programmers to progress beyond this task analysis to develop customized programs. Readers should refer to Table 2 if difficulties arise during the programming process.

OVERVIEW OF THE FUNCTIONAL ANALYSIS PROGRAM

The current application will consist of two computer screens, which we will design. These screens are called *forms* in programming language. The first form will serve as an interface to allow the user to enter all of the relevant information for a functional analysis, including the condition, the length of the session in minutes, the name of the participant, the name of the observer, and a choice of observation method (interval recording and

frequency recording). The second form will contain control buttons to record the occurrence of up to three behaviors of interest, and in the case of interval recording, will prompt the user when to observe the participant and when to record behaviors.

In the current application, we will create an interface that will allow data to be collected on up to three behaviors (suggestions to allow more behaviors to be recorded are provided later). To individualize this recording system, the data-collection form will read in the behaviors of interest from a text file specific to the participant of the analysis. The form will also write the resulting observations to a text file that can be opened in a spreadsheet or statistical program such as Microsoft Excel or SPSS for graphing and analysis.

The current application was designed to use both frequency and interval recording methods. Fortunately, all interval recording procedures (e.g., whole, partial, or momentary) rely on the same built-in functionality; the only difference exists in the instructions given to the user of the application. For example, the user would be instructed to record an occurrence of the target behavior if it occurred during any part of the observation interval for partial-interval recording, if it occurred throughout the entire observation interval for whole-interval recording, and if it occurred at the end of the observation interval for momentary time sampling.

CREATING FORMS AND CONTROL OBJECTS

Starting an Application in Microsoft Visual Basic

Once the user has successfully installed the Visual Studio 2005 or Visual Basic 2005 programming language on his or her computer, the software can be opened like any other application. To begin creating the application, open Microsoft Visual Studio 2005, click on the ''Start'' icon in the lower left of the screen, and select ''All Programs.'' Find ''Microsoft Visual Studio 2005'' and click the icon to start the program. From the ''Recent Projects'' box in the upper left of the screen locate the ''Create:'' tab and click on the ''Project'' option. In the resulting window the user should locate the ''Visual Basic'' tab in the ''Project Types'' window. Beneath this tab is a subtab labeled ''Smart Device.'' The user should click on the + tab next to the smart device tab to reveal the different smart-device application templates available for use. The user should locate and click on the ''Windows Mobile 5.0 Pocket PC'' subtab. The different types of Pocket PC application templates available for use should be displayed in the ''Visual Studio installed templates'' window on the right. For this application, we will be creating a device application for the Pocket PC. In the ''Templates'' window, click on ''Device Application.'' This menu also allows the user to enter a name for the project in the textbox below the ''Project Types'' and ''Templates'' windows. For this application, type the name ''FA PDA'' and click the ''OK'' button in the lower right of the window.

Before moving on, save the program to prevent losing any work. In the upper left of the screen, locate and click on the ''File'' tab. From the resulting options locate and click on ''Save All.'' This will reveal a window prompting you to choose a location to save the project. To the right of the ''Location'' box is a button labeled ''Browse.'' Click on ''Browse'' to open the project location window. For the current application we will create a new folder in the C drive. From the drop-down box to the right of ''Look in:'' choose ''Local Disk (C:)'' and click the new folder icon. Enter the name for the new folder as ''FA PDA'' and click ''OK'' to return to the save project window. Click ''Save'' in the lower right corner to return to the main interface.

You will now see the graphic interface of the program. In the main window of the program you will see four to five smaller windows based on how your display is set up. The top panel of

Figure 1 displays the graphical layout of the programming interface. If any of the windows described below are not present, they can be added by clicking on ''View'' at the top of the screen and selecting the needed window from the available options. On the left of the screen should be the ''Toolbox'' window that contains various control objects that can be added to your project. In the center of the screen should be a window called the ''Object Browser,'' with a box labeled ''Form 1.'' The form is essentially the display that will be projected on the screen of the Pocket PC with which the user will interact, and consists of both a graphical or ''Design'' level and a syntax or code level. One window is located directly below ''Form 1'' with tabs titled ''Error List'' and ''Task List.'' The ''Task List'' and ''Error List'' will show errors and warnings that will prevent the application from functioning as they occur during programming. Two more windows should appear to the right of ''Form 1.'' These windows are named ''Solution Explorer—FA PDA'' and ''Properties.'' The solution explorer contains a file folder entitled ''FA PDA'' and serves as a visual directory of all of the contents of the project. The properties window lists a variety of properties of the selected form or control object including such things as size, name, and color.

Control Objects

As described above, the ''Toolbox'' window contains control objects such as buttons, timers, and textboxes. that can be added to forms. For the current project we will need to use several different types of control objects (described below).

We will need to use a control object that will allow the user to input text through the use of the Pocket PC's built-in keyboard, or soft input panel. This can be accomplished with a ''TextBox'' control. On the first form we will need to include three textboxes to allow the user to input the name of the participant, the name of the observer, and the length of the session.

We will also need to use a control object called a ''ComboBox.'' A combobox is a drop-down box that allows the user to choose between a collection of options. For the current project we will use two comboboxes to allow the user to select the condition being studied and the type of recording method to be used.

We will also need to use control objects that will visually orient the user to the purposes of these items. To accomplish this, ''Labels'' can be used to instruct users as to what needs to be typed or selected from each control object.

For this project, we will also need to use several ''Button'' controls. Buttons are used to carry out some specified task when the user clicks on the button with the Pocket PC's stylus. We will need to use one button on the first form to move from the first to the second form after all of the relevant information has been entered. On the second form, we will need three buttons to record the occurrence of target behaviors. We will also need one button for error correction and one button to start the observation period.

The final type of control for this application is a timer. A timer is an object that will perform some specified task at the end of some specified interval of time. For this project we will use three timers. One timer will be used to end the overall observation session. Two more timers are needed for interval recording to signal observation and recording periods and to write the results of each interval to the output file. We can now move on to creating the needed forms.

Creating the Program Forms: Renaming Form 1

Purpose and rationale. To start the project we need to create the two forms described above. To begin, we will change the name of Form 1. No specific conventions exist for naming forms or items within a project, but a good rule of thumb is to choose a name that specifies the function of an item. In this case, this form will be the first that will be encountered when running the application, so we will name it ''frmStart.'' Whenever a form's name is changed,

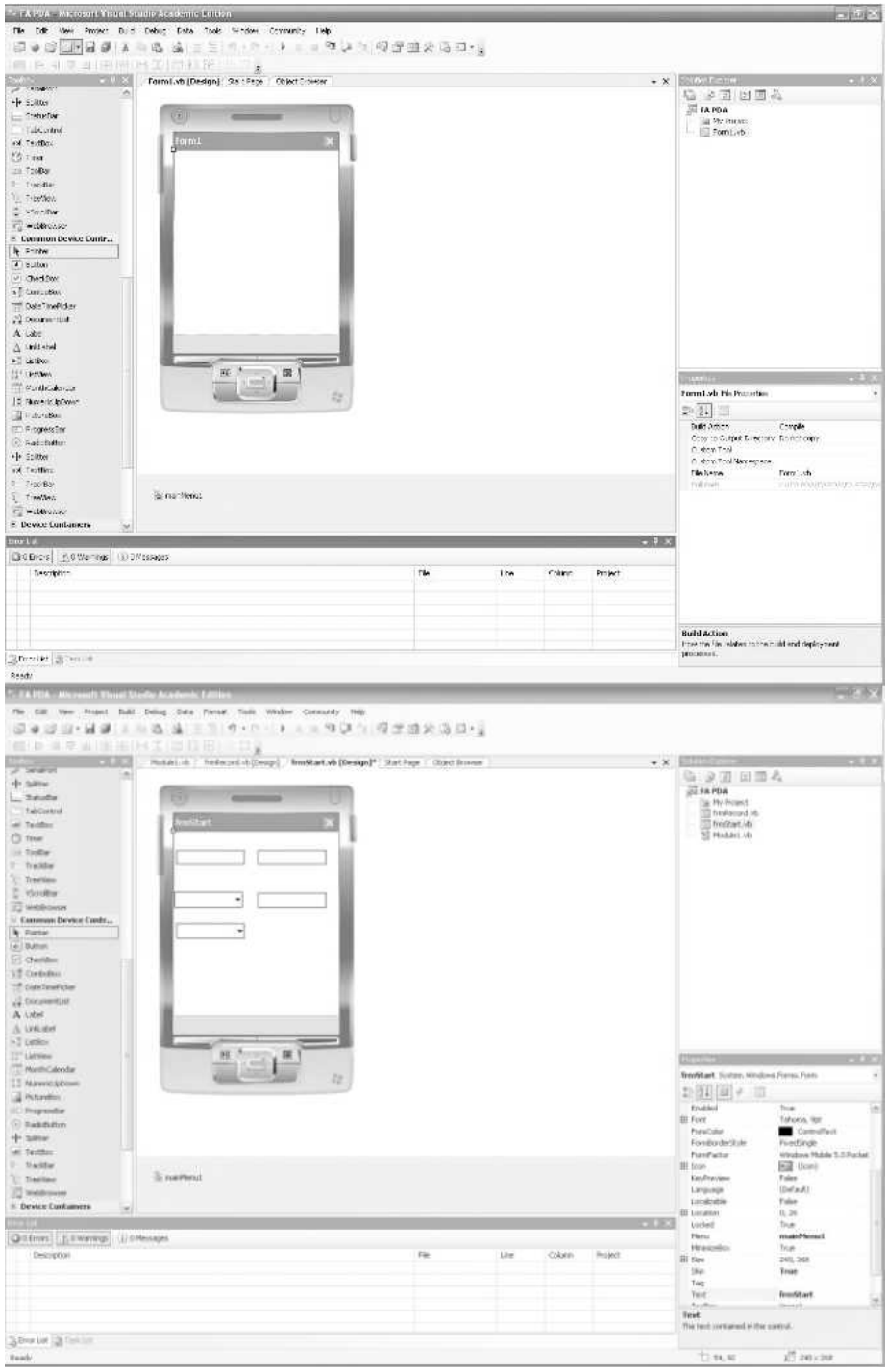

Figure 1. The initial computer programming interface (top) and initial user interface setup for the data-entry form (bottom).

it must be changed under both the properties window and the solution explorer window to ensure the program works properly. Other properties besides the name of the form can also be changed, including the form's color or text property. The text property is what will be seen at the top of the form when it is running on the Pocket PC. Currently the text setting should read "Form 1," so we will change the property to match the form's new name.

Prompts and actions. To select Form 1, click on it once. Under the properties window locate the "(Name)" option. Highlight the text, and hit delete. Under the "(Name)" option type "frmStart." Use your mouse to highlight "Form1.vb" in the solution explorer window. Click the right mouse button, choose the "Rename" option, type "frmStart.vb" and hit enter. To change the text property, highlight the form in the main window, then locate the "Text" property in the properties window. Highlight the current text, hit delete, and type "frmStart."

Creating the Program Forms: frmRecord and Module 1

Purpose and rationale. We will also need to create a second form that will be used to record instances of the target behaviors. Following the naming conventions described above, we will name this new form "frmRecord." Another item is needed to link the two forms to pass variables and events between the two forms. This can be accomplished by using a module.

Prompts and actions. With your mouse, highlight the project "FA PDA" in the solution explorer window. Click the right mouse button and move the mouse pointer over the "Add" option. When you highlight "Add," several options should be presented. In this case, we wish to add another windows form, so locate this option and click on it to open the "Add New Item" window. In this window, you will be given the option of choosing a name for the new form and choosing a template for the new item. Make sure that "Windows Form" is highlighted in the "Templates" window. In the name box, type "frmRecord.vb," and click "Add." The text and name properties should already be set as "frmRecord" for the new form, but the user may wish to adjust other properties such as the form's border style, but for now make sure the form's color is left as the default setting of white.

To add a module to the project, use your mouse to highlight the project "FA PDA" in the solution explorer. Click the right mouse button and move the mouse pointer over the "Add" option. From the options listed click on "Add Module." In the resulting new item window you can choose a name for the module or leave the name as "Module 1." Click on "Add" to complete the addition.

Adding the Control Objects to the Forms

Purpose and rationale. The next step in creating the application is to add all of the controls with which we will need to interact from the toolbox on the left of the screen. To add objects from the toolbox to the form, the user can click on the type of object needed and then click on the form. Once the object has been added to the form, the user may drag it to the desired location and resize it to the desired dimensions using the mouse. We will begin adding items to frmStart.

As stated previously, we will need to use two comboboxes to allow the user to select the condition being examined and the type of recording method used. Once added, we will change the comboboxes' name and items properties. We will also use three textboxes to allow the user to input the participant's name, the observer's name, and the length of the observation session in minutes. Once added, we will change the textboxes' name and text properties. We will also include five label objects to orient users to the purpose of each control. Once added, we will change the labels' Name and Text properties. The final item we will need for frmStart is a button control to allow the user to advance to the recording form

once all of the relevant information has been entered. Once added, we will need to change the button's name and text properties.

Prompts and actions. Across the top of the object browser is a horizontal list of all of the components of the current project. Locate and click on "frmStart.vb [Design]." In the toolbox menu to the left of the screen locate the combobox object. Add the object to the form as described above. Once the combobox has been added the user can adjust the size and location of the box either by using the mouse to drag it to the desired parameters and location or by adjusting the size and location settings in the properties window. When one clicks on an object on the form, the properties window will change focus from the form to the object in question.

The user should locate the "(Name)" option for the combobox in the properties window and highlight the existing text "ComboBox 1." Following naming conventions described above, the user should delete the text and type "cbCondition." The next step for this combobox is to add a collection of options corresponding to the conditions for which observations will occur. To add these options, locate and click on the "Items" property and click on the box to the right of "(Collection)."

The user will now be presented with the string collection editor into which the functional analysis condition options can be entered. The conditions we will use for this application will be attention, demand, alone, tangible, and play. The user should click on the center of the editor and type "attention," hit enter, type the next condition "demand," and so on, until each condition has been entered, then click "OK" at the bottom of the window.

Locate the textbox item in the toolbox to the right of the screen and add it to the form as described above. In the properties window, locate the "Name" option and change the textbox's name to "txtSession." The user should now locate the "Text" option and delete the text "Textbox 1," so that the text property is blank. Add a second textbox as described above, change the name property to "txtSubject," and locate the text property and delete the current text so that the property is blank. Add another textbox as described above, change the name property to "txtObserver," and set the text property to blank.

Add another combobox as described above and change the name property to "cbRecord." Locate the "Items" option in the properties window, and click the box to the right of "(Collection)" to open the string collection editor for the new combobox. With the mouse, click on the center of the editor and type "frequency," hit enter, type "partial interval," type "whole interval," type "momentary time sampling," and click "OK" along the bottom of the window. The form should now have two comboboxes and three textboxes. The bottom panel of Figure 1 displays the progress so far.

In the toolbox, locate the "Label" object, and add one to the form. Position the first label directly above "cbCondition," change the name property to "lblCondition," and the text property to "FA Condition." Add a second label, position it directly above "txtSession," and change the name property to "lblSession." Change the text property to "session length (min)" so that users will know that session times should be entered in minutes. Add a third label and position it directly above "txtSubject." Change the name to "lblSubject" and the text property to "subject." Add a fourth label and position it directly above "txtObserver." Change the name property to "lblObserver" and the text to "observer." Add a fifth and final label directly above "cbRecord," change the name property to "lblRecord," and change the text to "recording method." See the top panel of Figure 2 for ideas on how to position all of the objects.

In the toolbox to the left of the screen, locate the "Button" object and add one to the form. As described for other controls, change the

name property to ''butStart'' and the text property to ''start.'' You may wish to experiment with repositioning items on the form, changing text and font properties of the labels and textboxes to suit your tastes. The top panel of Figure 2 displays a final layout of the graphical interface of frmStart.

First Test of Program

Before moving on to adding needed objects to the second form, test to ensure that the layout works properly. Prior to this test, change the project's settings so that the program begins with frmStart. With your mouse, locate and click on ''Project'' at the top of the screen. Select ''FA PDA_Properties …'' and in the resulting window locate the ''Startup Object'' drop-down box. Click on the down arrow of the box and select ''frmStart.'' Click on ''Apply'' at the bottom of the window and click on ''OK.''

One of the strengths of developing applications in the compact framework is the option of testing programs with the Pocket PC Emulator. This ''virtual Pocket PC'' allows a test run of the program before transferring it over to an actual handheld device. The emulator is useful; however, due to compatibility problems with certain systems and the eventual need to transfer the program to the Pocket PC, we will describe how to test this application using an actual Pocket PC and Microsoft ActiveSync to connect the Pocket PC to the computer (the standard connection software included with all Pocket PCs).

First, be sure that the Pocket PC is plugged into its docking cradle and that the cradle is plugged into the computer. Second, make sure that Microsoft ActiveSync is running. If it is not already running, open Microsoft ActiveSync by using the mouse to click on the ''Start'' button in the lower left corner of the computer. Click on ''All Programs'' and locate and click on ''Microsoft ActiveSync.'' Return to the open instance of the application on your computer. At the top of the screen, locate and click on ''Debug.'' Click on the ''Start'' option. A window labeled ''Deploy FA PDA'' should have opened. In this window you are given the option of running the application in the Pocket PC emulator or on the actual Pocket PC. The emulator is the default setting, but in this instance, run the program on the actual Pocket PC. Click on ''Windows Mobile 5.0 Pocket PC Device'' and click on ''Deploy.''

The application should now be running on the Pocket PC sitting in its cradle. With the stylus, tap on the ''FA Condition'' combobox and make sure that all of the conditions entered previously are available. Next, tap on the ''Session Length (min)'' textbox. In the lower right corner of the screen should be an icon that looks like a miniature keyboard; tap on this box to open the keyboard. The keyboard works much like any other keyboard, with the exception of using the stylus instead of your fingers to strike buttons. Be sure that all of the textboxes are positioned high enough on the form that they are not obscured by the open keyboard. However, it doesn't matter if the ''Start'' button is obscured because by the time the user needs to tap it, all of the necessary text will have been entered, and the keyboard can be minimized. By contrast, if any of the textboxes are obscured, they will need to be repositioned. Be sure that you can enter text into all of the textboxes and that all of the options entered previously are available for the recording method combobox. To minimize the keyboard, click on the keyboard icon in the lower right corner of the screen. Return to the open instance of the application on your computer. At the top of the screen, locate and click on ''Debug'' and click on ''Stop Debugging.'' If any of the textboxes were obscured by the keyboard, reposition them as needed and run the test again.

Adding Controls to the Recording Form: Buttons

Purpose and rationale. On this form we will need a method to allow users of the application to start observation periods and to record specific behaviors. To accomplish this, we can

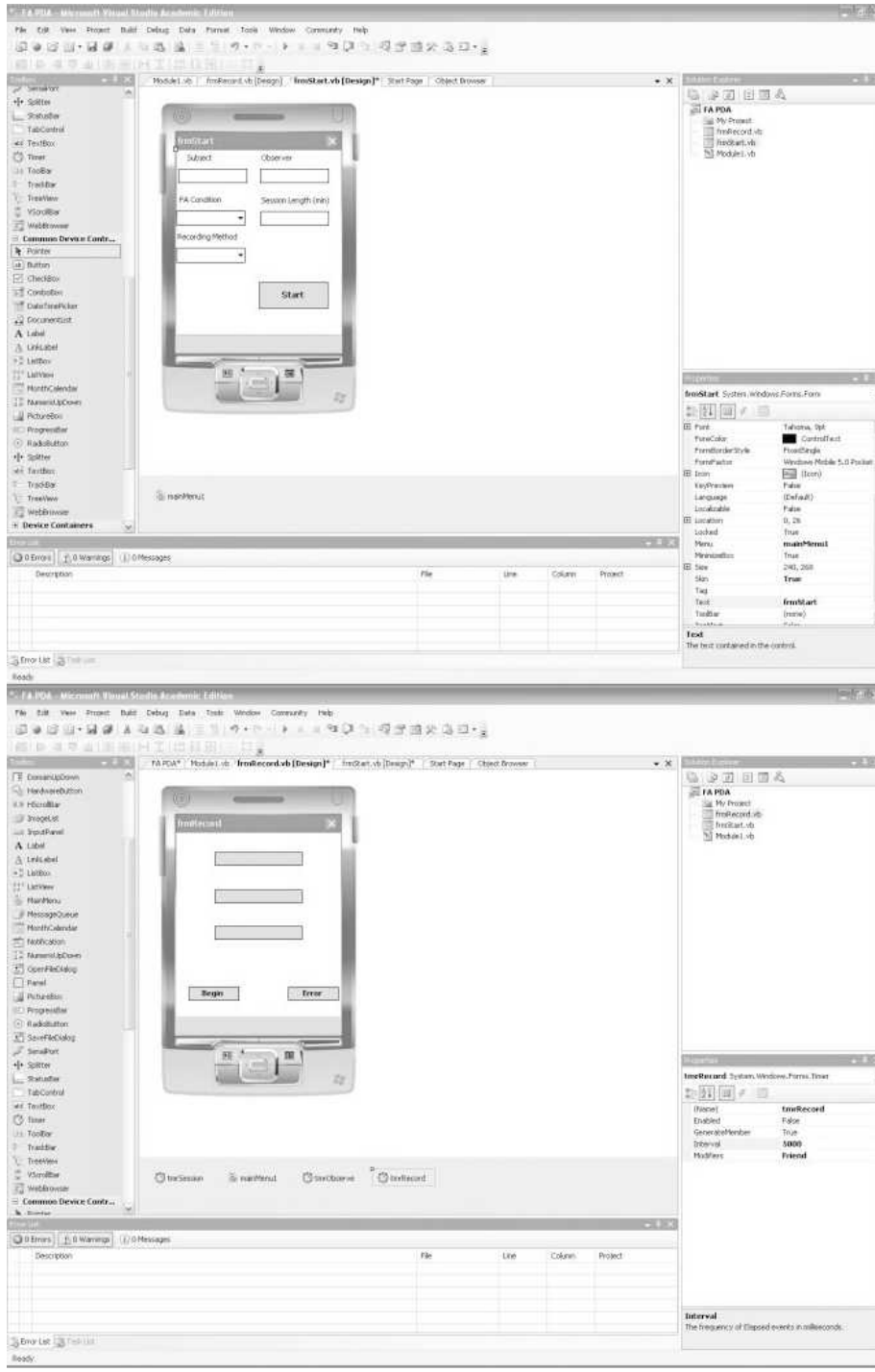

Figure 2. The final graphical setup for the first form (top) and the initial user interface for functional analysis data collection (bottom).

use a series of buttons. For this application, we want to build in the capability to observe and record up to three specific behaviors for a given participant. To do this we will need three buttons corresponding to the three possible behaviors.

At times, users will require more or fewer buttons corresponding to target behaviors. Fortunately, each button can be made visible or hidden. To hide buttons set the "Visible" property to false. This will make the objects invisible when the program is running unless they are needed for a given participant.

As described earlier, we will use a text file to input the behaviors in question to the program. The program will read the behaviors from a text file and display those behaviors as the text on the buttons. Buttons need to be wide enough for the text corresponding to the behavior, or the behavior must be abbreviated to fit on the button. To accommodate most behavior categories, the size property of the behaviors recording buttons should be set to 128, 20. We will also need one button to start the observation period and one button for error correction.

Prompts and actions. Along the top of the object browser, locate and click on "frmRecord. vb[Design]." Add three buttons to the form, and change the names of the buttons to "butBeh1," "butBeh2," and "butBeh3." Locate the "Visible" property for each button and change it to false. Set the size property of the behavior buttons to 128, 20. Locate the "Text" property for each button and delete the current text. Add another button, change the name to "butBegin," and change the text to "Begin." Add another button, change the name to "butError," and change the text property to "Error."

Adding Controls to the Recording Form: Timers

Purpose and rationale. As stated previously, we will need three timer controls. One timer will be used to end an observation session after the desired number of minutes selected on the first form, and the other two timers will be used for observation and recording intervals for interval recording. You will note that when a timer is added to the form, it immediately hops off the form and sits directly below it. This is fine, and is done because the timer is really not an object that anyone sees. We will then have to address three properties on each of the timers. The name of each timer will be changed to correspond with its function. The "Enabled" property corresponds to whether the timer is on or off. Because we want the timer to operate only when the program tells it to, we will make sure the enabled property is set to false for all three timers. The timer's interval property corresponds to how long it runs before performing its specified actions (in milliseconds). The interval property for the overall session timer will be set by the user by entering the session length on the first form, but we will set the interval properties for the observation and recording period timers. The length of observation and recording intervals is the prerogative of the person making the observation, but for the sake of this application we will use 10-s observation intervals and 5-s recording intervals.

Prompts and actions. Locate the timer object in the toolbox on the left of the screen, and add three timers to the form as described for other objects. For the first timer, change the name property to "tmrSession" and make sure the "Enabled" property is set to false. Change the name of the second timer to "tmrObserve," and set the enabled property to false. The interval property is scaled in milliseconds, so change the interval property of "tmrObserve" to 10000. Change the name of the final timer to "tmrRecord," set the enabled property to false, and the interval to 5000. The bottom panel of Figure 2 displays your progress so far with the interface of "frmRecord."

WRITING CODE FOR DATA COLLECTION

In this section, we will detail how to create the variables necessary and how to develop the

necessary syntax to allow the controls on the forms to be functional. At times throughout the rest of the paper, we will provide further description of the purpose and rationale of a specific area of code after the reader has been prompted to enter the code on the application.

Declarations: Public Variables

Purpose and rationale. Variables are used to keep track of information. In this application, some of the information we need to keep track of includes the information relevant to the functional analysis including the participant's name, the condition, the type of recording method, the session length, the number of specific target behaviors for a given participant, and information relating to the occurrence of the target behaviors. In the Visual Basic 2005 programming language, variables must be declared based on the type of information they are used to track. Variables that track nonnumeric or text data can be declared as "String" variables, and variables that track numeric data can be declared as "Integer" variables. The level of the variable is also an important consideration. If the variable is used to carry information across multiple forms, it must be declared as a "Public" variable in a module. For variables that track information specific to only one form, they may be declared on that form only.

For the current application, all of the information entered on the first form will need to be carried to the second recording form; therefore, declare variables related to the information as "Public" variables on the module that we created earlier. We will declare public "String" variables to track the condition, the participant's name, the name of the observer, and the type of recording method used. We will also declare public numeric or "Integer" variables for the length of the session and the type of recording method used.

Prompts and actions. Across the top of the object browser locate and click on "Module1.vb" to open the module. Under the text "Module Module1" and above "End Module" type

```
Public vCondition, vSubject, vObserver, vRe
    cordMethod As String
Public vSession, vRecord As Integer
```

Code for frmStart: Setting Values for Public Variables and Opening frmRecord

Purpose and rationale. In the syntax for frmStart, we will provide the information to set the values for the variables we just declared. The following code will allow the programmer to create a subroutine that will be performed whenever the user of the application clicks on the "Start" button. When the user clicks on the "Start" button, we want to set the values of the previously declared variables to the information entered into all of the textboxes and comboboxes on the form. We also want the program to close the first form and open the recording form.

Prompts and actions. Across the top of the object browser, locate and click on "frmStart.vb [Design]" to return to the first form. Now double click on the "Start" button. This will open the syntax or code level of the form. Initially the cursor will be flashing underneath the text "Private Sub butStart_Click()." Directly below the text "Private Sub butStart_ Click ()" and above "End Sub" type

```
vSubject = txtSubject.Text
vCondition = cbCondition.Text
vObserver = txtObserver.Text
vRecordMethod= cbRecord.Text
If cbRecord.Text = "Frequency" Then
   vRecord = 1
Else
   vRecord = 2
End If
vSession = txtSession.Text
Dim × As New frmRecord
x.Show ()
Me.Hide ()
```

Code for frmRecord: System and Form Variable Declarations

Purpose and rationale. For the current application, the program will read in the behaviors of interest from a text or ".txt" input

file that we will create. The output will also be written to a separate text file. For any Pocket PC application that will read from or write to a text file, a system declaration must be made that will allow the program to use what are called ''StreamReaders'' and ''StreamWriters.''

In the following code we will make the system declaration described above. We will also declare two numeric variables for tracking the occurrence of target behaviors. One variable will be used to track the occurrence of target behaviors during individual intervals for interval recording, and one will be used to track the cumulative total of the target behaviors across the whole session for both frequency and interval recording. We will also need a few variables for information retrieved from the input files for a given participant. We will declare a numeric variable for the number of target behaviors specific to an individual participant. We will declare string variables for the text of those behaviors. We will also need to declare numeric variables for tracking errors and for tracking the number of intervals during the session for interval recording. Finally, we will declare an instance of a ''StreamWriter'' that will open and write to an individual participant's output file.

Prompts and actions. To begin, locate and click on ''frmRecord.vb [Design]'' in the object browser to get back to the recording form. Double click on a blank area of the form. This should take you to the code level of the form in the ''Private sub frmRecord_Load ()'' subroutine. With your mouse, scroll to the top of the screen, click before ''Public Class frmRecord,'' and hit enter. On the resulting blank line at the top of the screen, type

```
Option Explicit On
Imports System.IO
Imports System.Text
```

Directly below the text ''Public Class frmRe cord'' type the following lines of code:

```
Dim vBeh1, vBeh2, vBeh3, vTotalBeh1, vTo-
    talBeh2, vTotalBeh3 as Integer
Dim vBehaviors as Integer
Dim beh1, beh2, beh3 as String
Dim vError, vInt as Integer
Dim swBeh As StreamWriter = New IO.Stream-
    Writer (''FA Data\Output Files\'' & vSub-
    ject & vCondition & ''output.txt'')
```

CREATING AN INPUT FILE AND FILE FOLDERS ON THE POCKET PC

Before adding any more code, we will describe how to create an input file for an individual participant and how to set up a file structure on the Pocket PC for storing input and output files. A simple way of creating an input file is to use a simple text editor such as Microsoft Notepad. The user should click on ''Start'' in the lower left of the computer screen and move the mouse pointer over ''All Programs.'' Notepad can be found under the ''Accessories'' tab for all computers running Windows XP. Locate and click on ''Notepad'' to launch the program.

Let's look at a hypothetical participant, John Doe. John has a history of displaying three types of self-injurious behaviors including head banging, hair pulling, and skin gouging. In Notepad, type the number of behaviors of interest, hit enter, type the first behavior, hit enter, type the second behavior, hit enter, and so on until all of the behaviors have been listed. For John the user should type

```
3
Head Banging
Hair Pulling
Skin Gouging
```

Now click on ''File'' at the top of the screen and choose ''Save As.'' In the resulting window go to the ''Save In'' drop-down menu and choose the desktop. We will be moving this file to the Pocket PC later and need to make it easy to find. In the file name box, the user should type ''John Doe.txt,'' and click ''Save.'' The user can now close Notepad.

The first thing we need to do before we can test the program again is to develop the

necessary file folders on the Pocket PC for placing the input and output files. Make sure the Pocket PC is plugged into its docking cradle and that the cradle is plugged into the computer. If it is not already running, open Microsoft ActiveSync by using the mouse to click on the ''Start'' button in the lower left corner. Click on ''All Programs'' and locate and click on ''Microsoft ActiveSync.'' Now click on ''Explore'' in the upper middle of the Microsoft ActiveSync window. In the resulting window, click on ''My Windows Mobile-Based Device.'' The user should now click the right mouse button and select the ''New Folder'' option. Hit the ''delete'' key to delete the text ''New Folder'' and type ''FA Data.'' Double click on the just-created ''FA Data'' folder.

We now need to create two subfolders for input and output files. Click the right mouse button and select the ''New Folder'' option. Delete the current text and type ''input files.'' To create the second folder, click the right mouse button and select the ''New Folder'' option. Delete the current name and type ''output files.''

We now need to place the ''John Doe'' text file created earlier in the ''Input Files'' folder on the Pocket PC. Return to the computer's desktop where we saved the ''John Doe'' file. Highlight the file, click the right mouse button, and choose the ''Cut'' option. Now return to Microsoft ActiveSync. Click on ''Explore,'' and in the resulting window click on ''My Windows Mobile-Based Device.'' Double click on the ''FA Data'' folder, then double click on the ''Input Files'' folder. Click the right mouse button, and choose the ''Paste'' option. We can now move on to describing the code needed to read from this file.

Writing Code to Read from a Text File

Purpose and rationale. The following code will be used to read an individual participant's target behaviors from the text file that we just created. The code will open an instance of a ''StreamReader'' that will open the participant's input file and read the specific number of target behaviors for that client. The StreamReader will also read the text of those behaviors, place that text on the recording buttons, and make the proper number of recording buttons visible for the given participant. Because different participants may have fewer than three target behaviors, the following code will use what is called a ''Select Case'' statement to perform all of the needed actions based on the specific number of target behaviors.

Prompts and actions. Return to the open instance of the application on the computer. (Please note that in the following code and throughout the rest of the task analysis, an underscore preceding a line of code indicates that the code belongs on the same line as the code above but was separated due to the format of the article.) Type the following lines immediately below ''Private sub frmRecord_ Load ()'' and above ''End Sub'':

```
Dim srBeh as StreamReader = New IO.Stream
    "_" Reader("FA Data\Input Files\" & vSubject
    "_" & ".txt")
 vBehaviors = srBeh.ReadLine
 Select Case vBehaviors
    Case 1
       butBeh1.Visible = True
       beh1 = srBeh.ReadLine
       butBeh1.Text = beh1
       srBeh.Close ()
    Case 2
       butBeh1.Visible = True
       beh1 = srBeh.ReadLine
       butBeh1.Text = beh1
       butBeh2.Visible = True
       beh2 = srBeh.ReadLine
       butBeh2.Text = beh2
       srBeh.Close ()
    Case 3
       butBeh1.Visible = True
       beh1 = srBeh.ReadLine
       butBeh1.Text = beh1
       butBeh2.Visible = True
       beh2 = srBeh.ReadLine
```

```
    butBeh2.Text = beh2
    butBeh3.Visible = True
    beh3 = SrBeh.ReadLine
    butBeh3.Text = beh3
    srBeh.Close ()
End Select
```

Second Test of Program

Before moving on to the rest of the code, test the program again and make sure that all of the behaviors load properly. Make sure the Pocket PC is plugged into its docking cradle, the cradle is plugged into the computer, and Microsoft ActiveSync is running. As described previously, locate and click on ''Debug'' at the top of the screen and select ''Start.'' Choose ''Windows Mobile 5.0 Pocket PC Device'' and click on ''Deploy.''

The application should now be running on the Pocket PC sitting in its cradle. With the stylus, tap on the ''FA Condition'' combobox and choose the ''Attention'' condition. Tap on the ''Session Length (min)'' textbox. Open the keyboard by tapping on the icon in the lower right corner of the screen. To test the program, we will arrange brief 1-min sessions. Tap on the 1 on the keyboard, tap on the ''Participant'' textbox, and, using the stylus and keyboard, enter the name ''John Doe'' (make sure that the case of the letters and spelling match the file name placed in the ''Input Files'' folder). Tap on the ''Observer'' textbox and enter your name. Minimize the keyboard by tapping on the keyboard icon in the lower right corner of the screen. Tap on the ''Recording Method'' combobox and choose the ''Frequency'' option. Tap the ''Start'' button to move on to the data-collection form.

You should now see the data-collection form with three buttons for recording the specific behaviors for ''John Doe'' and the ''Begin'' and ''Error'' buttons. If everything loaded properly, the three specific behaviors should be displayed on the recording buttons. We can now return to the open instance of Visual Basic on the computer, locate and click on ''Debug,'' and click on ''Stop Debugging.'' The top panel of Figure 3 displays your completed graphical interface.

DEVELOPING FUNCTIONAL ANALYSIS PROGRAM CODE

Writing Code for Recording the Occurrence of Target Behaviors and Error Correction

Purpose and rationale. We will now create the necessary code for tracking the occurrence of the target behaviors and for tracking and correcting errors during the data collection process. During observation periods, the user of the application should be able to track the occurrence of target behaviors by clicking on the recording buttons. If the user makes a mistake, he or she should be able to subtract one occurrence of the mistaken behavior by tapping on the ''Error'' button then by tapping on the behavior he or she wishes to correct. We will describe the following code more thoroughly after it has been entered.

Prompts and actions. The user should return to the design level of ''frmRecord'' by locating and clicking on ''frmRecord.vb [Design]'' at the top of the main window. On the form, locate and double click on the first behavior recording button, ''butBeh1.'' This should return the user to the code level of the form and create a new subroutine titled, ''Private Sub butBeh1_Click ().'' In this subroutine, type

```
If vError = 1 Then
   vBeh1 = 0
   vTotalBeh1 = vTotalBeh1- 1
   If vTotalBeh1 < 0 then
      vTotalBeh1 = 0
   End If
   butBeh1.Enabled = True
   Me.BackColor = Color.White
Else
   vBeh1 = 1
   vTotalBeh1 = vTotalBeh1 + 1
End If
If vRecord = 2 Then
```

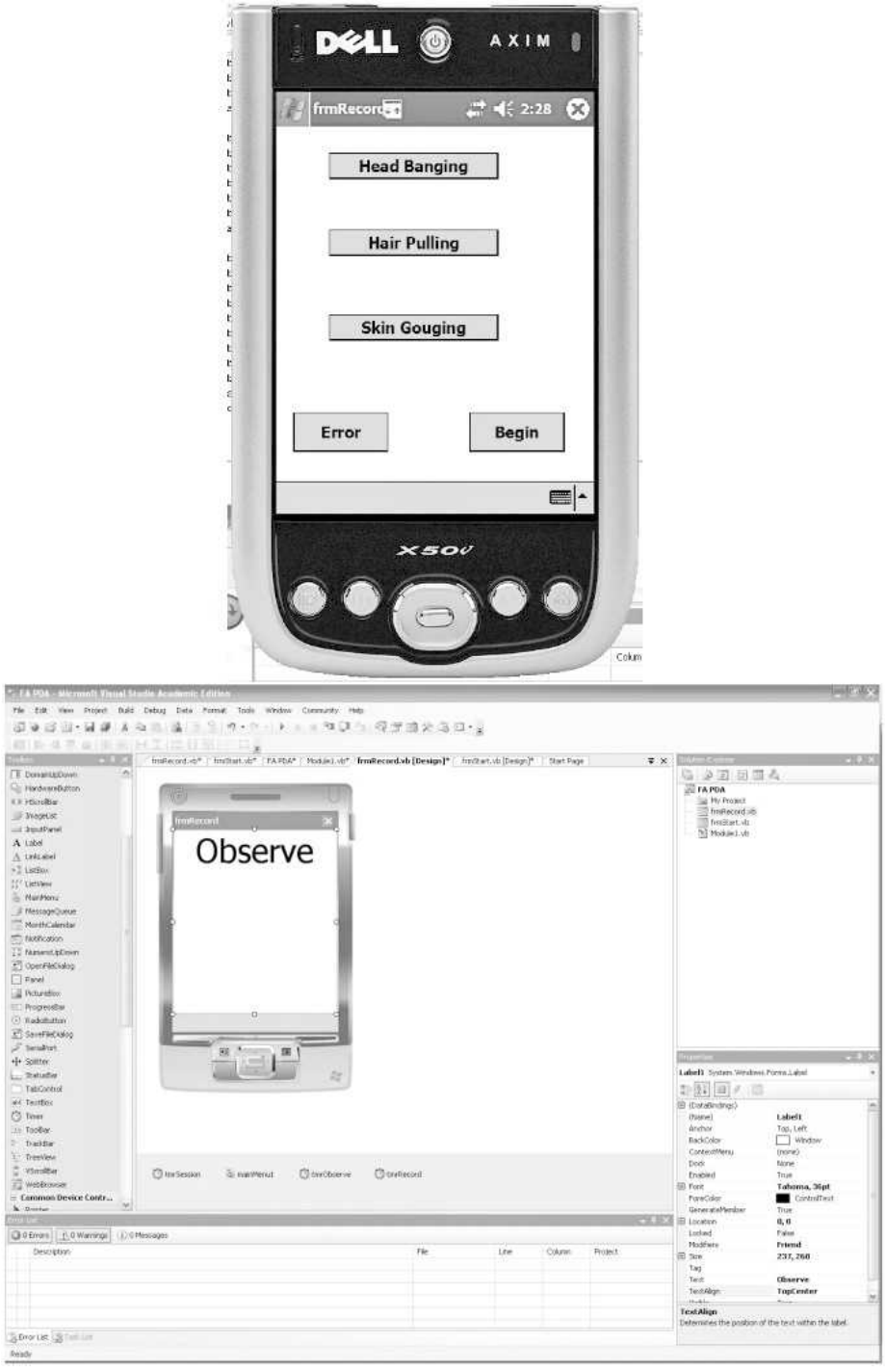

Figure 3. A run-time graphic of the frmStart in which users can input the various data-collection parameters (top) and the addition of a large ''observe'' label that will cover all other buttons during the final programming steps (bottom).

```
    butBeh1.Enabled = False
  End If
  vError = 0
```

Now return to the design level of the form as described previously. Locate and double click on the second button, "butBeh2," to create a subroutine titled "Private sub butBeh2_Click ()." The user can save time and effort by copying and pasting the code for the "Private sub butBeh1_Click ()" subroutine into the new subroutine. With the mouse, highlight the code in the "Private sub butBeh1_Click ()" subroutine, click the right mouse button, and choose "Copy." Click your mouse below "Private sub butBeh2_Click (()," click the right mouse button, and choose "Paste." In the resulting code, replace all 10 instances of the text "Beh1" with "Beh2." The user should now repeat this process for the remaining button ("butBeh3").

Return to the design level of the form as previously described. Locate and double click the "Error" button, which will create the "Pub sub butError_Click ()" subroutine. In this routine, type

```
vError = 1
butBeh1.Enabled = True
butBeh2.Enabled = True
butBeh3.Enabled = True
Me.BackColor = Color.Red
```

Writing Code for Starting Observation Periods

Purpose and rationale. We can now focus on the code necessary for starting observation sessions. The user of the application will start observation periods by clicking on the "Begin" button. When the user clicks on this button, we want that button to become invisible because it is no longer necessary. Previously, we set the interval properties for the observation and recording interval timers for interval recording. We now need a line of code to set the interval property of the overall session timer, "tmrSession," to the value the user entered on the first form. Because the interval property for timers is scaled in milliseconds and the value entered for the session time on the first form was scaled in minutes, we will need to multiply that value by 60,000. Once the interval timer is set we need to turn on the overall session timer by setting its "Enabled" property to true. If interval recording is the recording method of choice, we also need to turn on the observation timer and signal to the user that he or she should be observing the participant.

Prompts and actions. Directly below "Private Sub butBegin_Click ()" and before "End Sub," type the following lines of code:

```
butBegin.Visible = False
tmrSession.Interval = (60000 * vSession)
tmrSession.Enabled = True
If vRecord = 2 Then
  lblHide.Visible = True
  tmrObserve.Enabled = True
End If
```

The line of code "lblHide.Visible = True" should have generated an error visible in the Error List window below the form. The error should read "Name 'lblHide' is not declared" and occurred because as of yet there is no object on the form named "lblHide."

During interval recording we need an object to signal to the observer when to observe the participant and when to record instances of behavior. Return to the design level of the form as described previously. Add a label to the form and using the mouse resize and position the label so that it covers all of the buttons on the form. Change the name of the label to "lblHide," change the text property to "Observe," and change the visible property to false. It may be beneficial to enlarge the size of the text on the label. Locate the font property and click on the text. This should produce a box to the right of the text, similar to what was seen with the combobox's "Items" property previously. Click on the box and in the resulting window change the size property to 36 and click "OK." The bottom panel of Figure 3 displays the addition of lblHide.

Writing Code for the Output File: The Header

Purpose and rationale. When the observation begins, we want the program to begin writing to

the output file for the given participant. The first bits of information we want the program to write to the output file are a few header lines that report all of the relevant information of the functional analysis, including the observation's date and time and the details of the functional analysis entered on the first form.

At this time, we also want the program to write the behaviors of interest to the output file. We will have it write all of the behaviors on a single line of text separated by commas. This is done so that later when the output file is imported into a statistical program such as Microsoft Excel, the line of text can be used as header for the behaviors of interest. This process will be discussed more thoroughly in a later section. Because different participants will have different numbers of behaviors of interest, we need to use another select case statement. When interval recording is used, we also need the interval to be written into this statement. To accomplish this, if … then statements within the select case statement will be used.

Prompts and actions. Locate and click on "frmRecord.vb*" along the top of the object browser to return to the code level of the form. Under the last line of code in the "Private Sub butBegin_Click ()" subroutine (e.g., "End If') and before "End Sub," type

```
swBeh.WriteLine (DateTime.Now)
swBeh.WriteLine ("FACondition=, " vCondition)
swBeh.WriteLine ("Subject=, " & vSubject)
swBeh.WriteLine ("Observer=, " & vObserver)
swBeh.WriteLine ("Session Length=," & vSess-
    ion)
swBeh.WriteLine ("RecordingMethod=, " &
    vRecordMethod)
swBeh.WriteLine("# of Behaviors=," &
    vBehaviors)
Select Case vBehaviors
  Case 1
    If vRecord = 2 Then
      swBeh.WriteLine ("Interval, " & beh1)
    Else
    swBeh.WriteLine (beh1)
    End If
  Case 2
    If vRecord = 2 Then
      swBeh.WriteLine ("Interval, " & beh1
          "_" & "," & beh2)
    Else
    swBeh.WriteLine (beh1 & "," & beh2)
    End If
  Case 3
    If vRecord = 2 Then
      swBeh.WriteLine ("Interval, " & beh1 &
          "_" "," & beh2 & "," _& beh3)
    Else
    swBeh.WriteLine(beh1 & "," & beh2 & ","
        "_" & beh3)
    End If
End Select
```

Writing Code for Timers: tmrObserve

Purpose and rationale. If interval recording is the recording method of choice, the observation timer will be turned on when the user hits the "Begin" button to start an observation session. We have currently set the interval to 10 s. At the end of those 10 s, we need the label we included to prompt observations—lblHide—to become invisible. We also need to enable all of the recording buttons at this time. We can also use this timer to cumulatively advance the variable we declared to track the number of intervals—vInt. We also need to turn off tmrObserve, and turn on the timer for recording behavior—tmrRecord.

Prompts and actions. Return to the design level of frmRecord as described previously. Double click on tmrObserve to return to the code level of the form. This should have created a new subroutine, "Private Sub tmrObserve_Tick ()." In this subroutine, type

```
vInt = vInt + 1
tmrRecord.Enabled = True
tmrObserve.Enabled = False
lblHide.Visible = False
butBeh1.Enabled = True
butBeh2.Enabled = True
butBeh3.Enabled = True
```

Writing Code for Timers: tmrRecord

Purpose and rationale. We now need to address what occurs when the recording timer, tmrRecord, times out. Currently this timer is set at 5 s. During that time, the user will record instances of behavior. At the end of those 5 s, the timer needs to turn off and the program needs to write the results of the interval to the output file.

Recall previously that the last bit of code we wrote in the subroutine for the "Begin" button was intended to write all of the behaviors of interest to a single line, separated by commas, in the output file. We want the results of each interval to be written the same way, in the same order, on one line separated by commas, so that when the file is imported into a spreadsheet program, the results of each interval will line up under the corresponding behavior. Due to the fact that different participants will have different numbers of behaviors, we once again need to use a select case statement.

After the program has written the results of the interval to the output file, it is ready to move on to the next observation interval. To progress, the program needs to reset the variables for tracking the occurrence of the target behaviors and errors back to zero, turn the observation timer back on, and prompt the user to begin observing the participant again by making lblHide visible again.

Prompts and actions. Return to the design level of the frmRecord and double click on tmrRecord. This should open the "Private Sub tmrRecord_Tick ()" subroutine. In the subroutine, type

```
Select Case vBehaviors
   Case 1
      swBeh.WriteLine (vInt & "," & vBeh1)
   Case 2
      swBeh.WriteLine (vInt & "," & vBeh1 & ","
         "_" & vBeh2)
   Case 3
      swBeh.WriteLine (vInt & "," & vBeh1 & ","
         "_" & vBeh2 & "," & _vBeh3)
   End Select
lblHide.Visible = True
tmrObserve.Enabled = True
tmrRecord.Enabled = False
vBeh1 = 0
vBeh2 = 0
vBeh3 = 0
vError = 0
Me.BackColor = Color.White
```

Writing Code for Timers: tmrSession

Purpose and rationale. We now need to address what the program does at the end of the entire session when the session timer, tmrSession, reaches the end of its interval. At the end of the session, all currently running timers need to be turned off. For frequency recording, we want the program to write the cumulative total of all behaviors to the output file. As described above, we want the results to be written to the output file on one line separated by commas. When interval recording is used, we want the program to calculate the percentage of intervals in which behaviors occurred. We also want the program to write these percentages to the output file on one line, separated by commas, in the order of the behaviors written to the output file when the user hit the "Begin" button. To accomplish these tasks, we will use two if … then statements corresponding to whether the recording method chosen is interval or frequency recording. Due to the fact that different participants will have different numbers of behavior, we will also have to use a select case statement within each if … then statement.

Prompts and actions. Return to the design level of frmRecord and double click on tmrSession. This should open the "Private Sub tmrSession_Tick ()" subroutine in the code level of the form. In this subroutine, type

```
tmrObserve.Enabled = False
tmrRecord.Enabled = False
tmrSession.Enabled = False
If vRecord = 1 Then
   Select Case vBehaviors
      Case 1
```

```
            swBeh.WriteLine (vTotalBeh1)
        Case 2
            swBeh.WriteLine (vTotalBeh1 "," &
                vTotalBeh2)
        Case 3
            swBeh.WriteLine (vTotalBeh1 "," &
                vTotalBeh2 & "," _& vTotalBeh3)
        End Select
    End If
    If vRecord = 2 Then
        swBeh.WriteLine ("Percentage of Intervals")
        Dim a, b, c as integer
        a = (vTotalBeh1 / vInt * 100)
        b = (vTotalBeh2 / vInt * 100)
        c = (vTotalBeh3 / vInt * 100)
        Select Case vBehaviors
            Case 1
                swBeh.WriteLine ("Intervals," & beh1)
                swBeh.WriteLine(vInt & "," a)
            Case 2
                swBeh.WriteLine ("Intervals," & beh1 &
                    "_" "," & beh2)
                swBeh.WriteLine (vInt & "," & a & "," &
                    "_" b)
            Case 3
                swBeh.WriteLine ("Intervals," & beh1 &
                    "_" "," & beh2 & "," _& beh3)
                swBeh.WriteLine (vInt & "," & a & "," &
                    "_" b & "," & c)
        End Select
    End If
    swBeh.Close ()
    Dim × As New frmStart
    x.Show ()
    Me.Close ()
```

Third Test of Program

Test the application for the final time on the Pocket PC. Make sure that the Pocket PC is plugged into its cradle, then locate and click on "Debug" at the top of the screen. Select "Start" and in the next window, select "Windows Mobile 5.0 Pocket PC Device" and click on "Deploy."

The application should now be running on the Pocket PC sitting in its cradle. With the stylus, tap on the "FA Condition" combobox and choose the "Attention" condition. Tap on the "Session Length (min)" textbox. Open the keyboard, tap on the "1" on the keyboard, tap on the "Participant" textbox, and enter the name "John Doe," making sure that the case of the letters and spelling match the file name we placed in the "Input Files" folder. Click on the "Observer" textbox and enter your name. Minimize the keyboard by clicking on the keyboard icon again in the lower right corner of the screen. Click on the "Recording Method" combobox, choose the "Frequency" option, and click on the "Start" button to move on to the data-collection form.

You should now see the data-collection form with three buttons for recording the specific behaviors for "John Doe" and the "Begin" button. For the test, we want to make sure all three recording buttons and the "Error" button are working properly. Click on the "Begin" button to start the observation session. With the stylus, tap each recording button six times. Tap the "Error" button, then tap the "Head Banging" button. Once again tap the "Error" button, then tap the "Hair Pulling" button. Once again tap the "Error" button, then tap the "Skin Gouging" button. After the end of the 1-min session, the application should return to the first form.

Check the output file and make sure everything was recorded correctly. Return to the open instance of the application on your computer. At the top of the screen, locate and click on "Debug" and click on "Stop Debugging." Now return to the Pocket PC. With the stylus, tap "Start" in the top left of the screen. Tap "Programs" and on the resulting screen, locate and tap "File Explorer." This should provide access to the "My Documents" folder, but the "FA Data" folder that we are looking for is located in the directory above this. In the gray area near the top of the screen, tap "My Documents" and in the resulting drop-down menu tap "My Device." On the resulting

screen, locate and tap the ''FA Data'' folder, then tap the ''Output Files'' folder. You should now see the ''John DoeAttentionoutput'' file.

Using the ''StreamWriter'' method of writing output files creates a problem that will appear the second time the user of this application wants to observe the same participant under the same condition. The way the ''StreamWriter'' works, each additional observation will write over the previous file. To prevent the loss of data, the user of the application can either rename or move the output file after each observation. To prevent overwriting this output file, tap and hold the stylus on the ''John DoeAttentionoutput'' file. In the resulting window choose the ''Rename'' option. On the keyboard, tap the right arrow, then add a 1 to the end of the file name and tap the enter key. This will prevent the file from being overwritten the next time a ''John DoeAttention'' observation is made.

To determine if everything was written to the output file correctly, tap on the file with the stylus. The file should have opened in the Pocket Word program. You should see several lines of text beginning with the date and time the observation was made, followed by the condition, the participant's name, the observer's name, the session length, the recording method, the behaviors of interest, and finally the frequency of each behavior. If everything worked properly, the frequency of all behaviors should be five, because we clicked each recording button six times and then with the ''Error'' button subtracted one instance of each behavior.

Check to be sure that the program works properly when partial-interval recording is the chosen recording method. With the stylus tap ''OK'' in the top right corner of the screen. Return to the application on the computer. As described above, start the program on the Pocket PC by locating and clicking on ''Debug'' at the top of the screen, then click on ''Start.'' In the resulting window, choose ''Windows Mobile 5.0 Pocket PC Device,'' and click on ''Deploy.'' Once the program is running on the Pocket PC, follow the instructions above for entering the relevant information for the session, only this time in the ''Recording Method'' combobox choose ''Partial Interval.'' Tap on the ''Start'' button to move to the behavior recording form. Click on the begin button to start the observation. The ''Observe'' label should appear, covering all of the buttons. After 10 s, the ''Observe'' label should disappear, and the buttons should be available for recording behavior. To test whether the application is working properly, at the end of the first observation interval, tap all three recording buttons. After the second interval, tap the ''Head Banging'' button. After the third interval, tap the ''Hair Pulling'' button. After the fourth interval, tap the ''Skin Gouging'' button. The program should return to the first form at the end of 1 min. Return to the application on the computer and stop debugging as described previously.

On the Pocket PC a new ''John DoeAttentionoutput'' file should have been written. Rename the file as described previously to prevent overwriting it on subsequent observations. Open the new output file as described previously. As stated above, the file should consist of the date and time of the observation on the first line, followed by the condition, the participant, the observer, the session length, the recording method, the number of behaviors, and the behaviors of interest. There should also be three lines corresponding to the first three intervals of the observation, with positive instances of behaviors noted with a 1 and absences noted with a 0. At this point you might be wondering where the results of the fourth interval are. A problem occurs in writing to the output file because the overall session timer (''tmrSession'') reaches the end of its interval slightly before the recording timer (''tmrRecord'') reaches the end of its interval during the last interval of the session. The data from the final interval are not actually lost. The data from the final interval are still counted in the overall session data, as can be seen by

looking at the last few lines of the output file. The program has calculated the percentage of intervals for the observation period that each behavior occurred. During the four intervals of the observation, we recorded two instances of each behavior, and the last line should demonstrate this by reporting that four intervals were observed, with each behavior occurring during 50% of those intervals. With the stylus, close the file by tapping "OK" in the upper right corner of the screen.

Including a Method for Exiting the Program

The final version of the program is almost ready to install to the Pocket PC. However, we have not yet included a method for exiting the application without using the computer to end debugging. This can be accomplished using the Pocket PC's "Switcher Bar." The switcher bar is an item that allows users to see all currently running applications and either switch between them or close them. If the switcher bar is not already visible in the blue area at the top of the screen, it can be added by clicking on the windows icon in the top left of the screen and selecting "Settings." On the resulting screen locate and tap on "System" along the bottom of the screen. On the next screen locate and tap on the "Switcher Bar" icon, then tap the X in the top right corner of the screen.

Creating an Installation File and Installing the Final Version of the Software on the Pocket PC

At this point we will create a file that can be used to install the final version of the software on any Pocket PC. To accomplish this we will create a "CAB" file. Return to the open instance of the application on the computer. At the top of the screen above the object browser, locate and click on the "File" menu item. From the resulting list choose the "Add" option and from the options given click on "New Project." This should open the "Add New Project" window. Under the "Project types:" box click on the + to the left of "Other Project Types" and click on the "Setup and Deployment" option. Under the "Templates:" window you should see an option called "Smart Device Cab Project." Make sure this template is highlighted, then in the "Name:" box, replace the text "SmartDeviceCab1" with "FA PDA Setup" and click the "OK" button.

This should return you to the current project with a few additions. The object browser should now be set to a new screen with the title "File System (FA PDA Setup)." From this screen you can set where in the Pocket PC's file structure the program will be installed. In the solution explorer you should now note the addition of another project beneath the FA PDA project for the setup file. Before we can actually build the CAB file, we need to set some properties for this new project. If you highlight the FA PDA Setup project by clicking on it in the solution explorer, you should see some new properties in the properties window. From here you can include the name of the software manufacturer and the product name, and set some options for the mimimum and maximum operating systems a device must use for the software to install correctly. We should not have to worry about any of these, so we will move on to some more important properties.

With the mouse right click on "FA PDA Setup" in the solution explorer and from the resulting options highlight "Add" and choose "Project Output." In the resulting options dialogue, select the "Primary Output" option, and in the "Configuration" drop-down box choose "Active" and click "OK." This should add all of the dependent files to the project.

Now look at the split screens in the "File System (FA PDA Setup)" page in the object browser. Now we make sure that a shortcut to start the program will be created in the Pocket PC's programs menu. Locate and right click on the text "File System on Target Machine." From the resulting "Add Special Folder" options, select "Programs Folder." Now highlight the "Application Folder" in the left pane by clicking on it. The contents of the

"Application Folder" should now be visible in the right pane. These contents should consist of a single file called "Primary Output from FA PDA (Active)." Right click on this file and choose the "Create Shortcut …" option. Right click on the newly created shortcut file and choose the "Rename" option. Delete the current text and type "FA PDA." Now click and drag the "FA PDA" file to the "Programs Folder" in the left pane.

We are now ready to create the new CAB file. Click on "Build" on the main menu. From the resulting options click on "Build FA PDA Setup." Watch the bottom of the screen for some text that will initially read "Build started …" then "Building …" When this text changes to "Build succeeded," the file has been constructed.

We will now locate the CAB file just constructed and place it on the Pocket PC. Before proceeding make sure that Microsoft ActiveSync is currently running and that the Pocket PC is connected to its cradle. Click on the "Start" icon in the lower left of the screen. From the options given, click on "My Computer," and in the next window click on "Local Disc (C:)." On the resulting screen, locate and double click on the "FA PDA" folder. This should take you to another screen with another "FA PDA" folder. Double click on the "FA PDA" folder. Locate and double click on the "FA PDA Setup" folder, and on the resulting page, locate and double click on the "Debug" folder. From the resulting files locate select the "FA PDA Setup.CAB" file. Click the right mouse button and select copy. Return to the open instance of Microsoft ActiveSync on the computer. Locate and click on "Explore." On the resulting page, double click on "My Windows Mobile-Based Device." Click the right mouse button and select "Paste."

Now return to the Pocket PC in its cradle. Tap on the Windows icon in the upper left of the screen. From the options listed, select "Programs" and in the resulting window, locate and tap on "File Explorer." This should take you to the "My Device" folder. If not, tap on the gray area at the top of the screen below the Windows icon and select "My Device" from the available options. You should see the file we just placed on the Pocket PC at the bottom of the screen. To install the final version of the program, tap on the file "FA PDA Setup." The file should install and may produce a box either stating that the program may not display properly because it was designed for a previous version of Windows Mobile software or that the manufacturer is unknown and cannot be verified. If all of your previous tests have functioned properly, ignore these messages and tap "OK."

Final Test of Program

We can now test the application for the final time. Remove the Pocket PC from its cradle. In the blue area at the top of the screen, tap on the Windows icon. From the resulting options, tap on "Programs" and locate the "FA PDA" icon. Tap on "FA PDA" to launch the application, and run the same test for partial-interval recording we previously completed. At the end of the observation period, tap on the switcher bar in the blue area of the top of the screen and select "Exit current program." Rename and check the output file as described previously, and if everything checked out, then the program is complete and ready for use. All that is required for using the application with actual participants is to construct input files as we did for "John Doe," and place them in the "Input Files" folder on the Pocket PC. Figure 4 displays a Pocket PC screen ready to collect data.

IMPORTING DATA INTO MICROSOFT EXCEL

The data from the output files can be quickly imported into Microsoft Excel. Readers should refer to Carr and Burkholder (1998) for how to create single-subject graphs in Microsoft Excel, but we will describe how to import the output files into Microsoft Excel. Return to the open instance of Microsoft ActiveSync on the

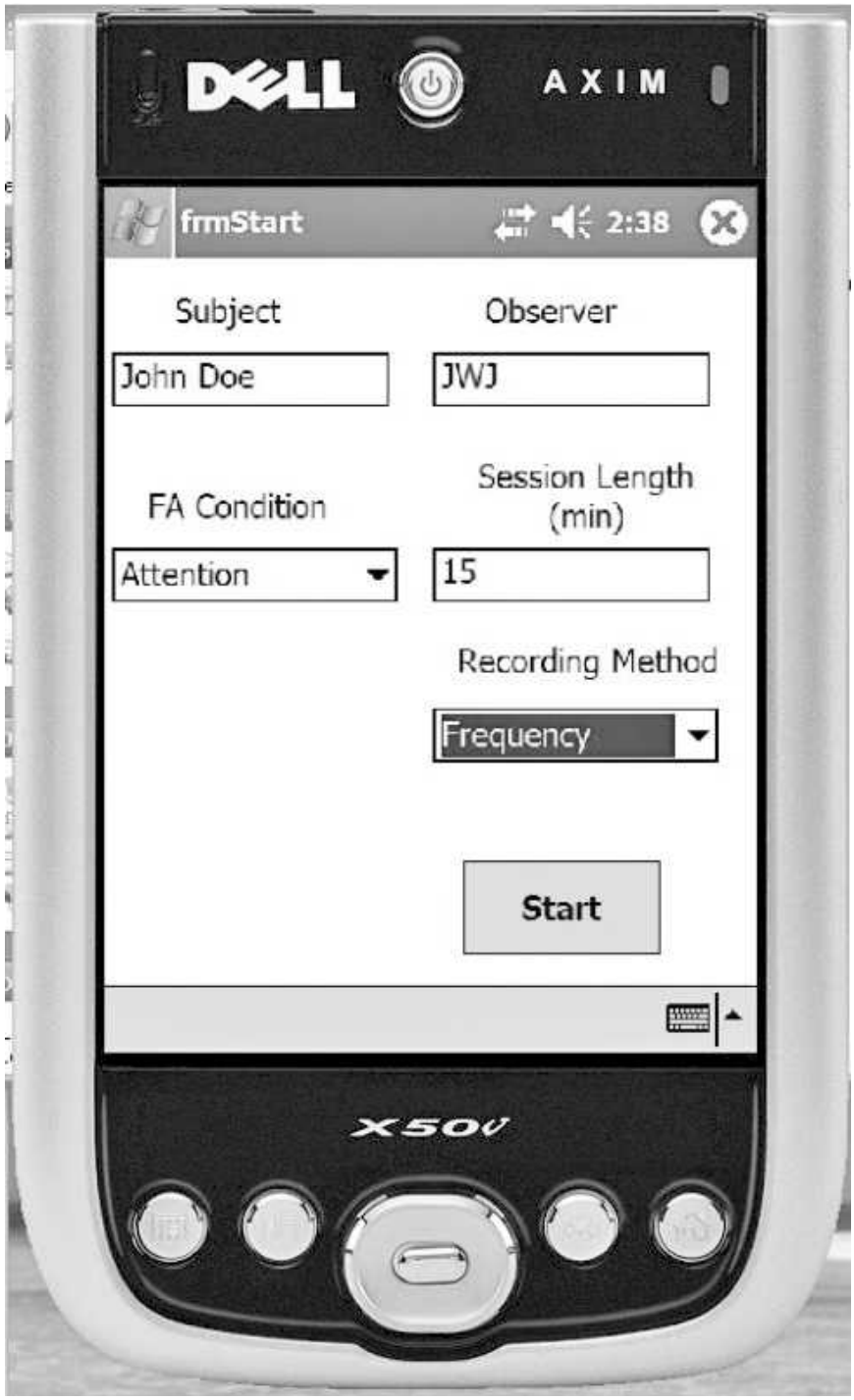

Figure 4. An actual Pocket PC screen with the programmed software installed and ready to collect data.

computer. Click on ''Explore.'' Double click on ''My Pocket PC,'' double click on the ''FA Data'' folder, and then double click on the ''Output Files'' folder. Highlight any one of the output files in the folder. Click the right mouse button and select ''Copy.'' You should now return to the computer's desktop, click the right mouse button, and select ''Paste.'' We now need to open Microsoft Excel.

Click on the ''start'' button in the lower left corner and click on ''All Programs.'' Locate and click on ''Microsoft Office,'' and then locate and click on ''Microsoft Excel.'' In Microsoft Excel, locate and click on ''File'' and then ''Open.'' In the resulting window, locate the ''Look in:'' drop-down box. Click on the box and change the focus to ''Desktop.'' You now need to locate the ''Files of type:'' drop-down box at the bottom of the window. Click on this box and from the options listed select ''Text Files.'' Now locate and highlight the output file that we placed on the desktop from the available files and click ''Open.'' You should now be presented with the ''Text Import Wizard'' window.

In the Text Import Wizard you are first presented with two options of how data are separated in the text file, either delimited or fixed width. In our example, we used commas to separate the data, so make sure the ''Delimited'' option is checked. Now click on ''Next'' in the lower right corner. In Step 2 of the Text Import Wizard, you will be presented with options for how data are delimited. Make sure that ''Commas'' is checked and click ''Finish'' in the lower left corner. The data should now be presented in spreadsheet format with all behaviors in separate columns ready to be graphed.

POSSIBILITIES FOR EXTENDING THE FUNCTIONALITY OF THE CURRENT PROGRAM

The current example consists of a very basic yet functional program for collecting functional analysis data using either frequency or multiple forms of interval recording. The intent of the current paper was to create a task analysis that allows a novice programmer to build a functional program and gain some experience that might aid him or her in developing a programming repertoire. There are many ways that the functionality and complexity of the current program could be extended, and readers may use some of the suggestions offered here to attempt to further their programming repertoire.

One potential area of flexibility in this program is the possibility of recording the occurrence of more than three behaviors. If one wished to construct an application with the capability of recording more than the three behaviors described here, one would need to simply include more buttons on the recording form and expand the ''Select Case'' statements used in the current application. The number of target behaviors is limited only by the number of buttons one can fit on the Pocket PC screen. One could also use the current program to track therapist's behaviors as well. Buttons could be included to track the delivery of attention, noncontingent reinforcement, physical contact, demands, and so on.

Another area of potential flexibility is the use of an auditory cue for observation and recording intervals. Although the use of sound files is a bit more difficult for Pocket PC applications than for applications for a full-size computer, ''wav'' files could be used to cue observation and recording intervals.

A user may also consider incorporating the use of separate observation and recording intervals. It is possible to eliminate the label that hides the buttons (lblHide) and combine the code for the observation and recording timers so that the results of individual intervals would be written to the output file at the end of a single interval. In this case, the user would record target behaviors as they occur, and the need for separate observation and recording intervals would be eliminated.

Another area of potential flexibility concerns the length of observation and recording inter-

vals. In the current example, they were set permanently at 10 s and 5 s, respectively, but it is also possible to allow the user the option of selecting other intervals. To accomplish this, the programmer could include textboxes or comboboxes on the first form to allow the user to input the intervals as we did for the overall session timer.

A final area of possible extension for the current program concerns the calculation of interobserver agreement. Although the current program automates the collection of actual real-time data, it does not automate the calculation of interobserver agreement. We have developed a separate application in Microsoft Visual Basic 2005 that is capable of reading the output files from the current application and calculating agreement for frequency recording, as well as both total and occurrence agreement for interval recording. This software can be downloaded free of charge from http://www.siu.edu/~rehabbat/IOA/IOAindex.html and includes step-by-step instructions for installation.

SUMMARY

Precise data collection, storage, and analysis are formidable tasks. Functional analyses in particular often require frequent observations of complex behavioral phenomena. Computer technology can aid in the collection of such data. The present paper described a means by which the process of data collection for functional analyses can in fact be computerized, and modifiable by the user for their own customized data-collection needs. The entire programming process requires about 4 hr, and a functional program along with a rudimentary repertoire of programming skills are the direct benefits of this investment. As more behavior analysts become aware of and capable of capitalizing on existing computer technologies, the more productive our field will become in creating solutions for socially important problems.

REFERENCES

Carr, J. E., & Burkholder, E. O. (1998). Creating single-subject design graphs with Microsoft Excel™. *Journal of Applied Behavior Analysis*, *31*, 245–251.

DeLeon, I. G., Arnold, K. L., Rodriguez-Catter, V., & Uy, M. L. (2003). Covariation between bizarre and nonbizarre speech as a function of the content of verbal attention. *Journal of Applied Behavior Analysis*, *36*, 101–104.

Dixon, M. R. (2003). Creating a portable data-collection system with Microsoft® embedded visual tools for the Pocket PC. *Journal of Applied Behavior Analysis*, *36*, 271–284.

Dixon, M. R., & MacLin, O. H. (2003). *Visual basic for behavioral psychologists*. Reno, NV: Context Press.

Erchul, W. P., & Martens, N. K. (2002). *School consultation*. New York: Springer.

Hanley, G. P., Iwata, B. A., & McCord, B. E. (2003). Functional analysis of problem behavior: A review. *Journal of Applied Behavior Analysis*, *36*, 147–185.

Iwata, B. A., Dorsey, M. F., Slifer, K. J., Bauman, K. E., & Richman, G. S. (1994). Toward a functional analysis of self-injury. *Journal of Applied Behavior Analysis*, *27*, 197–209. (Reprinted from *Analysis and Intervention in Developmental Disabilities, 2*, 3–20, 1982)

Kahng, S., & Iwata, B. A. (1998). Computerized systems for collecting real-time observational data. *Journal of Applied Behavior Analysis*, *31*, 253–261.

Received March 23, 2006
Final acceptance October 27, 2006
Action Editor, Gregory Hanley