CREATING AN IPHONE APPLICATION FOR COLLECTING CONTINUOUS ABC DATA

SETH W. WHITING AND MARK R. DIXON

SOUTHERN ILLINOIS UNIVERSITY

This paper provides an overview and task analysis for creating a continuous ABC data-collection application using Xcode on a Mac computer. Behavior analysts can program an ABC data collection system, complete with a customized list of target clients, antecedents, behaviors, and consequences to be recorded, and have the data automatically sent to an e-mail account after observations have concluded. Further suggestions are provided to customize the ABC data-collection system for individual preferences and clinical needs.

Key words: data collection, functional analysis, computers, iPhone

Current technology trends in the market data suggest that more than 50% of personal computing sales in 2011 will be mobile devices, including smartphones, tablet computers, and netbooks (Deloitte LLC, 2011). As technology advances and costs are reduced, more professionals within the field of behavior analysis likely will turn to mobile devices as a means to collect behavioral data. Jackson and Dixon (2007) and Dixon (2003) described simple task analyses in which a reader could develop frequency and interval data using a Pocket PC. In addition, MacLin, Dixon, and Jackson (2007) outlined steps to use a desktop PC to collect functional assessment data. Although these previously published reports still hold a fair degree of utility for today's behavior analysts, they use out-of-date software versions as well as the Windows Mobile Operating System that has dropped in market share from 12% to 6.8% from 2008 to 2010 (Gartner, Inc., 2008, 2010).

The Apple iPhone and iPod Touch initially released in 2007 have proven to be popular and user-friendly mobile devices. Over 73 million iPhones (Apple, Inc., 2010) and more than 45 million iPod Touches have been sold (Dediu, 2010) to date, and the availability of hundreds

of thousands of downloadable applications (i.e., apps) allow owners of these devices to enhance their phones with software that is relevant to their own individual needs. Several commercially available behavior-analytic apps are currently available for download from the Apple App Store. These programs provide simple means by which data can be collected and analyzed on the iPhone/iPod Touch. However, many of these currently available apps lack electronic data-transfer functions (to a database or spreadsheet) for subsequent analysis. Furthermore, commercial apps do not allow end users of the software to customize the layout or collection parameters to meet their own unique data-collection needs.

Although learning to program an iPhone app may seem like a daunting task, time spent recording and transferring observational data can be reduced once the app has been completed. The simplification of data collection with the use of a functional app may encourage observers such as direct care staff, parents, or teachers to collect more data. Observation with such a device could also prove to be less obtrusive and effortful than other means. The purpose of the present paper is to extend the literature on computerized data-collection development and provide the behavior analyst with a starting reference for basic iPhone/iPod Touch development for Xcode 4.0.2. The task analysis outlines the steps necessary to create a basic functional assessment data-collection sys-

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.2012.45-643

tem that records antecedents, behaviors, and consequences and automatically sends the data to any prespecified e-mail address in a format easily manipulated in spreadsheets and databases.

SYSTEM REQUIREMENTS AND SOFTWARE INSTALLATION

System Requirements, Registration, and Installation

An Intel Macintosh computer with at least six gigabytes of free hard-drive space is necessary to write software for the iPhone, iPod Touch, or iPad. Mac computers that run the Snow Leopard operating system (the minimum operating system required) are often available at university libraries or computer labs. If you wish to upload an app to a mobile device, the app may require that the newest version of the iPhone operating system to function on the device. The iTunes program offers frequent device updates for download and installation, and can be downloaded to a computer at <http://www.apple.com/itunes/download/>.

To download the software development kit to create apps, Apple requires each user to register as a developer. In your browser, navigate to <http://developer.apple.com/iphone/>. If you do not already have an Apple ID from another Apple service, you will need to click on the “Register” link. You will be prompted through the process, which includes creating an ID, answering several questions about your interests, and agreeing to the terms of use. Apple will send an e-mail to the target e-mail address with a link to activate your account.

Once you log in with your Apple ID, you will have access to free development content. With a free account, you will have access to all the tools necessary for development and will be able to run your apps on the iPhone simulator on your computer. However, if you’d like to join a paid developer program, you can click the “Register” link once again, and sign in with your existing account to upgrade. The standard paid account costs \$99 per year and allows the

user to upload created apps to a physical device, access the latest Beta versions of the software development kit when they are released, and distribute apps to the App Store for sale. There is also an option to join the Enterprise Program for \$299 per year to distribute apps to a large number of physical devices registered to a larger company, but you will not be able to upload them to the App Store for sale.

Whether you choose a free or paid account, you can download the software development kit by clicking on the “Xcode and iOS SDK” link. Once the files are downloaded, double-click on the iPhone SDK icon. The program will determine whether the computer is running the required operating system and install the software development kit to your hard drive. The installer will take you through the required steps, and will place a folder labeled “Developer” on your base hard drive.

Overview of Programming for the iPhone

The iPhone includes a wide range of functionality for use in development. The device can play music, display pictures and video, access high-speed internet, and respond to user input via the touch screen, motion, or tilting. However, apps that run on the iPhone are limited to screens that contain only one 320 by 480 pixel window at any given time, and use of the screen must be efficient for the best results.

The software development kit includes the necessary components to use the many functions of the iPhone. To write code for applications and lay out the logic the app will execute, you will use the Objective-C programming language. This is a user-friendly object-oriented programming language that employs commands that are typically easy to interpret. The commands used to make the iPhone respond to code in certain ways or interpret the various forms of input are called *classes*, and are defined in a subset of Objective-C functions termed *Cocoa Touch* elements. Cocoa Touch is a collection of frameworks that are responsible for building the created app on the iPhone and

executing the commands. You might consider Cocoa Touch a set of premade programming packages that give the iPhone its look and feel (e.g., the shape of the clock, the fading in and out of screen programs, the volume control display, etc.). All of these programming features are brought together in the *Xcode* program. Xcode is the actual software program in which you write the Objective-C programming syntax, or methods that define the behavior of the program, and call on the prepackaged scripts of Cocoa Touch. Finally, while working in Xcode, you can also access another subprogram called *Interface Builder* that allows the design of the application's appearance by placing objects (onscreen items capable of executing desired functions), make your app functional, and build the completed app. Because there is no single resource for behavior analysts who wish to begin development of data-collection systems on the iPhone, this paper serves as an initial resource. Additional materials will be needed to customize and expand beyond the task analysis offered here.

OVERVIEW OF THE FUNCTIONAL ASSESSMENT DATA-COLLECTION PROGRAM

The current iPhone application will consist of only a single screen, which will be designed by the user. What is shown on the screen is referred to as the *interface*. The interface will be designed to include spinning reels (termed a *picker*, similar to those on a slot machine). These reels contain all components necessary to quickly and easily record continuous ABC data in a clinical or applied setting, including a reel to select which client engaged in a target behavior and reels listing several common antecedents, target behaviors, and consequences. Along with this object, the screen view must contain labels corresponding to each reel to show what items are currently selected on each reel. Next, a text object is required to hold recorded data, and a button will allow a user to

enter information selected on the reels to be recorded. Finally, an additional button must be present to end the observational session and send all the recorded behaviors to the observer's e-mail address.

The current application will be written to include a reel that lists five clients, antecedents, target behaviors, and consequences. Each time data are entered into the text object, the information on each reel will be recorded and automatically followed by the time of entry and date. The data will be entered line by line and separated by commas so that the information can be taken out of the e-mail quickly and entered in to a spreadsheet program, such as Microsoft Excel or Numbers, if graphing or analysis is required.

DESIGNING THE VIEW CONTROLLER

Starting a Project in Xcode

After Xcode has been downloaded and installed from the Apple website and the user has created a profile (either paid or free), the software can be started similarly to any other application or program on the computer. Locate the folder on the computer labeled "Developer." From there, select the "Applications" folder and proceed to double-click on the Xcode icon.

Once Xcode has been opened, a welcome screen will appear from which the user can select among several options. Recently opened projects you have already started and previously saved can be opened on the right side of the window. On the left are options to create a new project, view a starting tutorial for beginners using Xcode, or access a developer website for more information. Click "Create a new Xcode project."

Next, the user can select a template. Be sure that, on the top left of the screen, "Application" is highlighted under "iOS" to make an app for the iPhone. The template we will use is the view-based application. This is a simple template that contains the components necessary to make an application with one screen.

Figure 1. The Xcode programming interface after starting a new project.

Proceed by clicking the “Next” button on the bottom. This window will prompt you to enter a product name (the name of your project) and a company identifier (an additional term with which to identify your work), and to select a device family where you indicate whether you want to develop an app for an iPhone, iPod, or iPad. For the purposes of this application, name the project “ABC Data” on the product name line, enter “testApp” on the second line, and make sure that the iPhone is the selected product. Leave “Include Unit Tests” checked. When you have selected the appropriate template and settings, click the “Next” button on the bottom of the window. The program will prompt you to create or designate a folder in which Xcode projects can be saved. Create a folder or select a preexisting folder in some location where you will easily be able to access it at a later time.

Now you will see the interface of Xcode. Figure 1 displays this graphical layout. The window is separated into three main components. The section on the left contains folders of all the files associated with the application being created. The main area on the right contains a panel that initially displays several options for the setup of your application, such as the screen orientation and other information regarding app building and deployment. Later, this larger area will be where code is entered. Finally, the very upper left portion of the screen contains a drop-down menu from which the user can select several build options as well as run and stop the debugger. If you have a free account with Apple, you must build your application on the iPhone simulator. Those who have purchased a paid account are eligible to upload iPhone apps directly to a registered iPhone or

iPod Touch by selecting “Device.” Make sure that “iPhone Simulator” is selected.

In the section on the left of the window, one of the five folders listed requires the user’s attention. The first folder listed is the “Classes” folder that contains most of the files you will be manipulating to make your app function. This folder has been given the same name as the project title. Click on the arrow on the left of the folder icon if it is not already expanded. This will cause each item contained in that folder to appear underneath. Here you will find a “.h” and “.m” file for the app delegate and another for each view controller in your project. Most of the development of an application will take place within these class files. The “.h” files are called *header* files. In the header files, all the methods and properties used by the corresponding class will be identified and listed. The “.m” file is called the *implementation* file. Here you can assign code to the methods and properties identified in the header file to make them behave in a specific way, such as react to screen touches or other user input. The second set of important files in this folder is the “.xib” files. These files are where you can run the interface builder and design what your application will look like on the iPhone screen.

Designing the View

In the same leftmost column, navigate down to the “ABC Data” folder and double-click on ABC_DataViewController.xib. This will cause the interface builder to display in the main screen. The view window shows what your application will look like on the iPhone screen. To start, the view should be a plain, gray rectangle. On the tab on the very top of the computer screen, select the “View” tab. From here, click on “Utilities,” and then select the “Object Library.” This will open a list of utilities for your app, and a set of all the objects that can be inserted into the view.

Objects required. Continuous ABC data recording requires selecting a client, antecedent, target behavior, and consequence. Further, each of these must have several available options

from which to choose. This can all be accomplished with one object in the library: a “Picker View.” This object can be customized to contain the four reels required for ABC data, and each reel can contain a large number of objects accessible with a simple scroll. “Label” objects that display text placed beneath the picker can show which items are currently selected on the picker view. This will promote accuracy of data entry because one will know exactly which item is selected.

The “Button” and “Text View” objects will be useful tools for entering an observed instance of behavior and holding those data until the observation is terminated. The button must transfer the information from the label objects that display the selected client, antecedent, behavior, and consequence to the text view object (a large container used to hold greater amounts of text) to store all the recorded data during the observation period. Finally, the last object required is a button that compiles all of the text entered throughout the observation period and prompts the iPhone to open an email program to send the data. This is a convenient way to take data off of the iPhone and transfer it to a spreadsheet or elsewhere in your records, as well as end the observation.

Placing objects on the view. On the bottom right of the Xcode, scroll down the list and find the “Picker View” object. Click on the object and drag it in to the gray view window. Be sure to center the object in the window so the whole picker view is visible. Note that this object’s width can be manipulated, but not its height. Place it on the very top of view. When adding objects to the view controller with interface builder, the picker view will always look the same; it will appear to have only one reel listing cities in California. This is normal. The properties of the picker view can be changed within the code written in the class files so that during run time it will show the desired amount of reels and values that you entered.

Once the picker view is on the view controller, navigate back to the objects library

Figure 2. The view of the ABC data-collection app in the interface builder after all objects needed to collect continuous ABC data have been added.

on the bottom right and locate the label object. Place four of them in a row directly underneath the picker view spaced across the length of the view controller. These will show the values currently selected on each reel in the picker view. To edit the default value of the label objects (the text the label will show when the application starts), click on the leftmost label, select the “View” tab from the top of the screen, point the cursor over “Utilities,” then click on “Attributes Inspector.” The right side of the Xcode window will display an area in which several features of the label can be adjusted. In this area, find the word “Text” followed by a box to enter text that currently shows the word “Label.” Delete the word “Label” and replace it with “Client.” Notice that the label on the view now shows the word

“Client.” In the same way, change the text in the next three labels to read “Antecedent,” “Behavior,” and “Consequence,” respectively. Resize the labels by clicking on each one and dragging the blue circles that appear around the object’s edges so that they are proportioned appropriately.

Click on the “Objects Library” window once again, and locate the “Round Rect Button” object. Below the labels, drag two buttons onto the view controller. Place the two buttons next to each other and stretch them horizontally so that they are large enough to be pressed by the user with ease. Click on the left button, and bring the attributes window to the front. This window should now read “Button” at the top. Editing the text in the text box following the word “Title” can change the text value of the button. Click on the text box and type the words “Enter Selection.” Change the text of the second button to “E-mail Data.”

The last object to add is the “Text View.” Once again, click on the “Object Library” window and locate the text view object. Drag a text view to the bottom of the view controller and stretch the object so that it fills most of the bottom of the screen. When placed in the view controller, the text view will be filled with a sentence written in Latin. In the “Attributes” window, delete all of this text, and replace it with the word “Data.” Adjust the alignment of the text to centered. Below where you typed the word “Data” is a check box followed by “Editable.” Make sure the box is unchecked, because you do not want the user to be able to type in the text view, but only enter observational data with the designated button that was previously placed on the view. When completed, the view should resemble that in Figure 2.

On the upper left corner of the Xcode window, click “Run.” This button should look like the “play” symbol typically used on audio software. The iPhone simulator will appear and show the view you have created. Except for the picker view that requires some code to appear, the view should now contain all the objects

required to record continuous ABC data. When you are finished, click the stop button on the Xcode window. If, on the left section of the screen, the folders window has been replaced with something else, the folder icon on the tab immediately below the run button can bring back the view of the folders.

Writing Code for ABC Data Collection

Using the picker view. As mentioned above, all objects placed in the view appeared when the application was run on the iPhone simulator except the picker view object. In order for the picker view to be visible during runtime, our view controller class must conform to several protocols. In the Xcode window, under the classes folder, click on “ABC_DataViewController.h.” The main window will now show the environment in which you will enter code. Some default code will already be entered in the code area. Add the bolded text to the “@interface” line so it reads:

```
@interface ABC_DataViewController :
    UIViewController
    <UIPickerViewDataSource,
    UIPickerViewDelegate> {
}
@end
```

The picker view must have multiple reels from which to select values. The data that will be held on each reel will be contained in an “NSArray.” The array must be populated with the items that will appear on each reel. Thus, one is required for clients, antecedents, behaviors, and consequences. Add the following code immediately beneath the “@interface” line:

```
NSArray *clientName;
NSArray *anteCedent;
NSArray *problemBx;
NSArray *conSequence;
```

This code declares four variables, each as an array that can hold multiple pieces of information. To define the values that each array holds, click on “ABC_DataViewController.m.”

Again, there will be default code already entered, except this time a lot of the font will be green. If any line is preceded by “//” or any block of code is preceded by “/*” and followed by “*/” it will be considered “marked out” and will turn green. If a segment of code is marked out, the code will not be read by the program during run time, and essentially will be skipped over. Scroll down the green text until you find the “viewDidLoad” method. Delete the “/*” and “*/” above and below the method, respectively. The font will return to its normal color, and the code will be active. Edit this method to read as follows:

```
- (void)viewDidLoad {
    clientName=[[NSArray alloc]
        initWithObjects:@"Bob", @"Pete",
        @"Julie", @"Stacey", @"Eric", nil];
    anteCedent=[[NSArray alloc] initWithObjects:
        @"Demand", @"Alone", @"Transition",
        @"FreePlay", @"Eating", nil];
    problemBx=[[NSArray alloc] initWithObjects:
        @"Slap", @"Spit", @"Bite", @"Pinch",
        @"Threat", nil];
    conSequence=[[NSArray alloc]
        initWithObjects: @"Attention",
        @"Ignored", @"Escape", @"Tangible",
        @"Redirected", nil];
    [super viewDidLoad];
}
```

Each of the arrays previously declared is listed, and the values that will appear on each reel of the picker view are defined. Note that each array always ends in “nil” to state that the array is concluded so that the application does not continue reading values that do not exist.

Now that reel values have been entered, we must use the “#define” directive to tell the application how many reels we want the picker view to contain. Scroll to the very top of the “ABC_DataViewController.m” code, and add the following code on top of the “#import” directive:

```
#define componentCount 4
#define clientComponent 0
```

```
#define antComponent 1
#define bxComponent 2
#define conComponent 3
```

“Component count” simply states how many total reels we would like the picker view to contain. Next, we list the name of each component as they appear on the picker view from left to right in numerical order starting with 0. We can refer to the various components that were previously defined by the declared names rather than the component number for easy identification in the future.

Back down below the “@implementation” line, two more methods need to be implemented. First, the “numberOfComponentsInPickerView” will return the value entered with the “#define” directive so the application will create a picker view with the identified number of components. Second, beneath the method just discussed, the “numberOfRowsInComponent” method must be implemented so that the application creates the necessary number of items on each reel. To do this, the program will count the number of items in each array that was defined earlier. Enter the following code beneath the “@implementation” below the ending bracket of the “viewDidLoad” method:

```
-(NSInteger)numberOfComponents
    InPickerView:(UIPickerView *)pickerView{
return componentCount;
}
-(NSInteger)pickerView:(UIPickerView
    *)pickerView numberOfRowsIn
    Component:(NSInteger)component{
if (component==clientComponent) {
return [clientName count];
}
if (component==bxComponent) {
return [problemBx count];
}
if (component==antComponent) {
return [anteCedent count];
}
if (component==conComponent) {
return [conSequence count];
}
```

Now, the layout of the picker view has been outlined, and the application can interpret the code to show the object with the intended dimensions. However, the values entered in the arrays will still not appear. To correct this problem, type the following code beneath the previously added methods:

```
-(NSString *)pickerView:(UIPickerView
    *)pickerView titleForRow:(NSInteger)row
    forComponent: (NSInteger)component{
if (component==clientComponent) {
return [clientName objectAtIndex:row];
}
if (component==bxComponent) {
return [problemBx objectAtIndex:row];
}
if (component==antComponent) {
return [anteCedent objectAtIndex:row];
}
if (component==conComponent) {
return [conSequence objectAtIndex:row];
}
}
```

This code enables the application to correctly identify the number corresponding to each row on the various reels and pull the appropriate item to display from the arrays that were defined earlier.

The last step required for the picker view to appear is to connect all the code that was just entered to the object in the interface. In the Xcode window under “ABC Data” folder, open the ABC_DataViewController.xib file once again. To connect the object to the code, hold down the control key, click on the picker view, and drag and release the mouse on “File’s Owner” icon on the small panel between the area with all the folders and the interface builder. The icon will look like an orange three-dimensional box. A small, dark box will appear. Click on “dataSource.” Next, control-drag again from the picker view to “File’s Owner”

and select “delegate.” Click “Run” in the Xcode window once again. The application should now run showing the picker populated with all the items entered in to the arrays.

Displaying selections in the labels. As the application is running, notice that longer words listed on the various reels are cut off. The width of the picker view is limited to the size of the iPhone screen and cannot fully display words over five or six letters. One solution for this problem is to use labels to display the full text of the items selected in the reels.

To begin, the labels must be declared in the `ABC_DataViewController.h` class file. Click on this file, and type the following code below where the arrays were declared:

```
IBOutlet UILabel *lblClient;
IBOutlet UILabel *lblBX;
IBOutlet UILabel *lblAnte;
IBOutlet UILabel *lblCon;
```

Using “IBOutlet” preceding the declaration of an object will allow that object to be recognized in interface builder so the code can be connected to the objects. Following the code just entered should be a closing bracket already in place. Beneath this bracket and before the “@end” line, set each label as a property by entering the following code:

```
@property (nonatomic,retain) UILabel
    *lblClient;
@property (nonatomic, retain) UILabel *lblBX;
@property (nonatomic, retain) UILabel
    *lblAnte;
@property (nonatomic, retain) UILabel
    *lblCon;
```

Proceed to the `ABC_DataViewController.m` class file, and insert the following code immediately under the “@implementation” line:

```
@synthesize lblClient;
@synthesize lblBX;
@synthesize lblAnte;
@synthesize lblCon;
```

By using the “@property” and “@synthesize” keywords, you will be able to use shorter, simpler code when referring to these objects.

Finally, the item selected on each reel in the picker must be displayed in the corresponding label below when it is selected. This is accomplished by working with the picker view and implementing the “didSelectRow” method. Enter the following method in the .m file where the other methods were previously entered:

```
-(void)pickerView:(UIPickerView *)pickerView
    didSelectRow:(NSInteger)row
    inComponent:(NSInteger) component{
    NSString *Clientmessage;
    NSString *BXmessage;
    NSString *anteMessage;
    NSString *conMessage;
    if (component==clientComponent) {
        Clientmessage=[[NSString alloc]
            initWithFormat:@"%@",
            [clientName objectAtIndex:row]];
        lblClient.text=Clientmessage;
    }
    if (component==bxComponent) {
        BXmessage=[[NSString alloc]
            initWithFormat:@"%@",
            [problemBx objectAtIndex:row]];
        lblBX.text=BXmessage;
    }
    if (component==antComponent) {
        anteMessage=[[NSString alloc]
            initWithFormat:@"%@",
            [anteCedent objectAtIndex:row]];
        lblAnte.text=anteMessage;
    }
    if (component==conComponent) {
        conMessage=[[NSString alloc]
            initWithFormat:@"%@",
            [conSequence objectAtIndex:row]];
        lblCon.text=conMessage;
    }
}
```

Figure 3. A build of the ABC data-collection app running in the iPhone simulator with the reels in the picker view populated with designated items and labels beneath displaying the values selected on each reel.

This code is executed when the reels on the picker view are manipulated by the user. First, several variables are declared within this method because they will only be used when this particular method is called. Second, when a row on a reel is selected, the value of that row is identified, set as a “string” variable, and placed in the label.

In order for these actions to occur during run time, open the same “.xib” file and form the appropriate connections between objects in the interface and the newly entered code. This

time, control-drag from “File’s Owner” to the label showing “Client.” Select “lblClient” from the options in the box. Perform this action for all remaining labels, connecting each label to its matching “lbl” name. Run the application in the simulator. The selections in the picker should now appear in the labels below as shown in Figure 3.

Storing data in the text view. Now that the labels display what the user selects in the picker view, the logic that will allow the user to store the data in the text view on the bottom of our application screen must be implemented. To start, declare the button object used to enter data and the text view object that stores the data in the ABC_DataViewController.h file. Enter the code that follows in to the “.h” file in the same location where the “UILabels” were previously declared:

```
IBOutlet UITextView *dataView;
IBOutlet UIButton *butEnterSelection;
```

Next, set both the button and the text view as properties by inserting the following code in the “.h” file where you set the “UILabels” as properties earlier:

```
@property (nonatomic, retain) UITextView
    *dataView;
@property (nonatomic, retain) UIButton
    *butEnterSelection;
```

The next step is to switch back to the ABC_DataViewController.m file, and synthesize the text view and button objects. Under the “@implementation” line where you synthesized the labels earlier, add:

```
@synthesize dataView;
@synthesize butEnterSelection;
```

The objects required to take the data from the labels to the text view are all set. This requires an “IBAction” method that will contain the logic to input the data in to the text view. Methods are declared in the same place as where objects are set as properties. Click on the ABC_DataViewController.h class

file, and enter the following code under the list of items beginning in “@property” and above the “@end” line:

```
-(IBAction)EnterSelection:(id)sender;
```

Click on the ABC_DataViewController.m file. We will now enter the code to specify what the method we defined in the header file will do. Most of the methods entered up to this point have involved the picker view. Locate the area where all the methods have been entered, and input the following code below them:

```
-(IBAction)EnterSelection:(id)sender;{
NSString *totalSelection;
NSDate *date1 = [NSDate date];
NSDateFormatter *dateFormatter =
    [[NSDateFormatter alloc] init];
[dateFormatter setDateFormat:@"
YYYY-MM-dd HH:mm:ss"];
NSString *myDate = [dateFormatter
    stringFromDate:date1];
totalSelection=[[NSString alloc]
    initWithFormat:dataView.text];
totalSelection= [totalSelection
    stringByAppendingString:@"\n"];
totalSelection= [totalSelection
    stringByAppendingString:lblClient.text];
totalSelection= [totalSelection
    stringByAppendingString:@" , "];
totalSelection= [totalSelection
    stringByAppendingString:lblAnte.text];
totalSelection= [totalSelection
    stringByAppendingString:@" , "];
totalSelection= [totalSelection
    stringByAppendingString:lblBX.text];
totalSelection= [totalSelection
    stringByAppendingString:@" , "];
totalSelection= [totalSelection
    stringByAppendingString:lblCon.text];
totalSelection= [totalSelection
    stringByAppendingString:@" , "];
totalSelection= [totalSelection
    stringByAppendingString: myDate];
dataView.text=totalSelection;
}
```

This method will perform a number of actions. First, two local variables are declared for use within the method: a string to hold the values from the labels and an “NSDate” which is a specialized string that will call the time from the iPhone. The “NSDateFormatter” is used to convert the time of day and date to appear in the format that we specified, and another string is declared to convert this to a data type that can be added in with the data from our labels. Next, the data are compiled. Starting with all the text already in the text view, “stringByAppendingString” adds information rather than replacing the current value of the string. The “\n” command starts the entry on a new line in the text view. Finally, all the information from the labels and the date are separated by commas and added to the total data. In short, everything in the text view is saved and removed, the new entry is added, and the text view refills with all of the data that have been collected.

The last step is to click on the ABC_DataViewController.xib file and form the necessary connections. Control-drag from the “Enter Selection” button to “File’s Owner” and select “EnterSelection” from the list that appears. Last, control-drag from “File’s Owner” to the text view object in the interface and select “dataView” from the list. When you run your application, it should now function similarly to what is shown in Figure 4.

E-mailing data. Click on ABC_DataViewController.h. Declare the last button object using the “IBOutlet” keyword. Formulating the e-mail requires two additional string variables to hold the recipient’s e-mail address and the information to be entered in to the e-mail’s subject line. Declare these variables and the button object in the “.h” file below the “@implementation” line among the arrays and other objects declared previously:

```
IBOutlet UIButton *btnSend;
NSString *txtTo;
NSString *txtSubject;
```

Figure 4. A build of the ABC data-collection app running in the iPhone simulator with observational data compiled in the text view.

Following the same pattern as before, set the new send button as a property:

```
@property (nonatomic, retain) UIButton
    *btnSend;
```

While you are still working in the header file, on the next line declare the following methods for e-mailing the data:

```
-(IBAction)send:(id) sender;
- (void) sendEmailTo:(NSString *)to
    withSubject:(NSString *) subject
    withBody:(NSString *)body;
```

When this is complete, switch to the “.m” file. Synthesize the button you just declared under the other synthesized objects by entering:

```
@synthesize btnSend;
```

Next, locate the last method you entered. On the line following your last method entry, enter the following code:

```
-(void) send:(id) sender {
    txtTo = @"YourEmail@yahoo.com";
    txtSubject = @"ABC Data";
    [self sendEmailTo:txtTo withSubject:txtSubject
        withBody:[dataView text]];
    [dataView resignFirstResponder];
}
```

This code sets the text that will be automatically entered as the recipient of the e-mail, the e-mail’s subject, and the e-mail’s body. The subject and recipient are defined here. Enter your own e-mail address following “txtTo = @” and the text you may want to appear in the subject line of the email in the “txtSubject = “ line. The body has been set to contain all the data that has been collected in the text view.

The last method organizes all the e-mail information and calls up the mail function of the iPhone. Below the last method entered, type the following code:

```
- (void) sendEmailTo:(NSString *)to
    withSubject:(NSString *) subject
    withBody:(NSString *)body {
    txtTo = @"YourEmail@yahoo.com";
    txtSubject = @"ABC Data";
    NSString *mailString = [NSString
        stringWithFormat:@"mailto:ff?to=%@
        &subject=%@&body=%@",
        [to stringByAddingPercentEscapes
            UsingEncoding:NSUTF8StringEncoding],
        [subject stringByAddingPercentEscapesUsing
            Encoding:NSUTF8StringEncoding],
        [body stringByAddingPercentEscapes
            UsingEncoding:NSUTF8StringEncoding]];
```

```

[[UIApplication sharedApplication]
  openURL:[NSURL
    URLWithString:mailString]];
}

```

The last step, just as before, is to open the `ABC_DataViewController.xib` file. Connect the last button you declared to the appropriate methods and code by control-dragging from “File’s Owner” to the “Email Data” button, and select “btnSend” from the list that appears. Then, control-drag from the same button to “File’s Owner” and click on “send:.”

Although the iPhone simulator is a great way to test your newly inputted code and test functionality as you go, it will not allow the user to send e-mail. This function is only available on a physical device (the “Email Data” button will appear to work correctly in the simulator). If you do load the app on to a physical device, the data will appear in the body of the e-mail when the “Email Data” button is pressed. On your computer, this can be easily loaded into Excel. Copy the data from the email and paste it to a “.txt” file (such as a Notepad file) and save. Open Excel, select the data tab on the top, click the “From Text” tool, and select the recently saved “.txt” file. From the window that appears, select “Delimited” from the top set of options, click the “Next” button, select a comma as a delimiter, click “Next,” and click “Finish.” The data should now be placed on the spreadsheet where it can be graphed or used to calculate conditional probabilities.

This completes all the design and code required to build a simple continuous ABC data-collection system. To make sure everything has been saved, highlight all of the files in the “ABC Data” folder, click “File” on the tab on top of your computer screen, and select “Save.” You can make several simple modifications to this program for further customization. First, the colors of the background or any object can be changed for a more attractive appearance. Next, object placement and size can be manipulated to promote easier interac-

tion. Also, the text you entered for the values that appear on the various reels can be customized to include names of your clients and common antecedents, behaviors, and consequences that you frequently observe. Finally, you can add more items to the reels by extending the arrays to include more values. In this way you can include every variable that you’re likely to encounter in your setting.

In summary, the ABC data-collection app built here can function as a useful tool for collecting continuous data on a number of clients in a clinical setting. The picker view allows an observer to record data quickly with only a few screen touches. Further, this app will hold a large amount of data for extended observations, because minimal code and few objects were required in the design of this app. Last, because each data entry includes the date and time the behavior occurred, the program could also be used over long amounts of time, and observations could be entered into a scatterplot to determine if behavior follows any pattern throughout the day.

Despite these advantages, the program, as written here, does have several limitations. The iPhone does have a memory limit, and the app will not function properly if this limit is exceeded. Although this occurs only after a very large amount of data has been entered, all observations can be lost potentially. Also, the values on the picker can be edited only prior to observation. If an unexpected behavior occurs or another client enters the area, the app cannot be updated during run time to input additional items on the reels to collect the necessary data. Finally, this app is intended to collect ABC data and cannot be used in all situations. With the basic skills outlined here, one can create other similar apps, such as frequency counters or duration timers, by combining the objects used here with adjustments to the methods employed and mathematical operations performed. Many of the objects available for use in iPhone programming function similarly to the objects used in the ABC data-collection app

and can be implemented successfully by following examples from Apple programming forums or the beginner's tutorials found on Apple's developer website.

REFERENCES

Apple, Inc. (2010, October 18). *Apple reports fourth quarter results* [Press release]. Retrieved from <http://www.apple.com/pr/library>

Dediu, H. (2010, September 3). *iPod Touch made up 37.7 percent of all iOS devices sold so far*. Retrieved from <http://www.asymco.com/2010/09/03/ipod-touch-made-up-37-7-percent-of-all-ios-devices-sold-so-far/>

Deloitte, LLC. (2011, January 20). *Deloitte analysis of top technology trends for 2011* [Press release]. Retrieved from http://www.deloitte.com/view/en_US/us/press/press-Releases

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.

Gartner, Inc. (2008, August). *Gartner says worldwide Smartphone sales grew 16 per cent in second quarter 2008* [Press release]. Retrieved from <http://www.gartner.com/it/page.jsp?id=754112>

Gartner, Inc. (2010, May). *Gartner says worldwide mobile phone sales grew 17 per cent in first quarter 2010* [Press release]. Retrieved from <http://www.gartner.com/it/page.jsp?id=1372013>

Jackson, J., & Dixon, M. R. (2007). A mobile computing solution for collecting functional analysis data on a Pocket PC. *Journal of Applied Behavior Analysis*, 40, 359–384.

MacLin, O. H., Dixon, M. R., & Jackson, J. W. (2007). *Visual basic 2005 for psychologists*. Reno, NV: Context Press.

Received February 21, 2011
Final acceptance May 9, 2012
Action Editor, Michael E. Kelley