

## Once upon a time there was a menu . . .

John Paul Elrod, Johns Hopkins University  
Marge Scerbo, University of Maryland at Baltimore

### INTRODUCTION

Menus are usually the first screens designed in an interactive system. Often, because basic menus are simple to develop, the original design is left unchanged while further and more extensive analysis and development on proceeding screens continues. It is important to remember that the menu is the first thing a user sees and its design might well affect how they perceive and use the system from the beginning. By taking a little extra time with the design of the system's menu screens, the final product may be far more user friendly and professional looking.

This paper follows the course of development of a SAS/AF system designed for the *NESUG Cafe*. The restaurant's Chef Hal and Manager Raymond contracted with JPM Consultants, Ltd. to design a computerized menu system that would allow their waitpersons to enter customer orders so that the information could be passed to kitchen staff and collected for further management analysis and reporting.

After an initial study of the Cafe's operations, the consultants identified many problems that could be corrected by the computerized system. The current system of processing orders relied on slips of paper that were passed to the kitchen. These slips were often misplaced, or misread by the kitchen staff causing incorrect and incomplete food preparation. The house specials changed daily and were prepared in limited quantities, and the staff did not always know what specials were available. The present system also makes analysis of orders and management reporting nearly impossible for Manager Raymond.

The consultants recommended that a network of computers be purchased and that the SAS System be used to implement the new menu system. The consultants decided that the computerized menu system should be structured like a paper menu and should be very easy to use in order to minimize the training needed for the Cafe's staff. Part of the paper menu is shown below.

| <i>NESUG Cafe</i>         |      |                 |      |
|---------------------------|------|-----------------|------|
| <i>Appetizers</i>         |      | <i>Desserts</i> |      |
| Shrimp Cocktail .....     | 4.00 | Chocolate ..... | 3.00 |
| Chicken Salad .....       | 2.50 | Apple Pie ..... | 2.75 |
| <i>Soups &amp; Salads</i> |      | <i>Drinks</i>   |      |
| Caesar Salad .....        | 3.00 | Coke .....      | 1.00 |
| Chicken Soup .....        | 2.75 | Coffee .....    | 2.75 |

### MENU ENTRIES

When the consultants first presented their findings to the Cafe, they also showed them a prototype menu built with a SAS/AF menu type entry. This type of menu took only a few minutes to develop and gave the customer a way of visualizing what the final menu might look like. The menu entry allows the user to select one choice from the menu list by typing a number on the command line at the top of the screen.

```

Select Option ==> _

                                NESUG Cafe

1 - Appetizers
2 - Soups and Salads
3 - Entrees
4 - Desserts
5 - Drinks
6 - House Specials
7 - Check Out

Please type your selection and press Enter.

```

The menu entry's only function is to control program flow by branching to another menu entry (sub-menu) or program entry. The menu entry consists of a display screen (shown above), a field attribute screen, and a general attribute screen. The field attribute screen defines which SAS/AF entry is displayed next when a selection is made. The field attribute screen for this menu entry looks like this:

| Option | Name   | Type    | Libref | Catalog |
|--------|--------|---------|--------|---------|
| 1      | appetz | menu    | *      | *       |
| 2      | sousld | menu    | *      | *       |
| 3      | entree | menu    | *      | *       |
| 4      | dessrt | menu    | *      | *       |
| 5      | drinks | menu    | *      | *       |
| 6      | specls | menu    | *      | *       |
| 7      | chkout | program | *      | *       |

The general attribute screen controls the general appearance and functionality of the menu entry. The only default general attribute that was changed for this menu was the banner attribute, which controls the appearance of the menu's command line. In this case, the consultants used the "select" option to display Select Option ==> at the top of the screen instead of Command ==>.

Although a menu entry is easy to create, there are

functional limitations with this entry type. No actual work can be performed by a menu entry; it can only branch to other entries. In addition, only one selection can be made. For these reasons, once the consultants were hired to create the system, menus were created exclusively with SAS/AF program entries.

## BASIC PROGRAM ENTRIES

To speed data input and eliminate the need to constantly return to the main menu, the system needed to be able to accept more than one selection at a time. This can be done by using a SAS/AF program entry instead of a menu entry. Program entries also provide for more flexibility in screen formats, eliminate the need for command lines, and allow the programmer full control over the application using Screen Control Language.

The display screen designed for the program entry was not too different in appearance from the menu type entry. The command or "selection" line at the top of the screen was replaced with selection fields underneath the menu text. These allowed for several choices to be entered at once. Each choice field was designated in Proc Build on the display screen by a single ampersand (&). If more than one choice is entered, the additional menus are displayed in sequence before returning to the main menu. The display screen for the program entry is shown below.

```

      NESUG Cafe

1 - Appetizers
2 - Soups and Salads
3 - Entrees
4 - Desserts
5 - Drinks
6 - House Specials
7 - Check Out

Please enter your choices: _ _ _
  
```

The command line was eliminated from the window by changing the banner attribute on the general attribute screen from the default COMMAND (which produces Command ==== at the top of the window) to NONE. Field attribute screens (one for each of the three "fields" or variables on the display screen) control what type of information can be entered into the field. In the attribute screen for each of these fields, they changed the field type from ACTION (the default for single character fields) to CHAR. Because each screen field is only one character long, the variable names could not be indicated on the screen, so each screen variable was assigned an alias in the field attributes screen. The alias is the variable name that is then used in the SCL program. They also entered > 1 7 in the list of valid values these fields will accept so

that the system will perform rudimentary error checking by automatically issuing an error message to the user if he/she chooses a number or letter not on the menu. Here are the important options on the attribute screen:

| Default Values     | Changed Values     |
|--------------------|--------------------|
| Field Name: FIELD1 | Field Name: FIELD1 |
| Alias: FIELD1      | Alias: CHOICE1     |
| Type: ACTION       | Type: CHAR         |
| List:              | List: > 1 7        |

In addition to the three parts of an entry found in a menu type entry, a program entry also has a source code screen associated with it that stores a screen control language program. The source code uses an entry statement to store the value of the customer's table number, which was passed to this program from a previous screen. It then creates an array for the three choice variables and performs a do loop to process each selection. The code uses a select statement to determine which selection was made, and branches to the proper sub-menu or program. The "call display" statement facilitates the branch to the next sub-menu and also passes the table number to the called program.

```
entry tableno $2;
array choice (3) choice1 - choice3
```

```
INIT:
return;
```

```
MAIN:
do i = 1 to 3;
  select(choice(i));
    when('1')
      call display("appetz.program",tableno);
    when('2')
      call display("sousld.program",tableno);
    when('3')
      call display("entree.program",tableno);
    when('4')
      call display("dessrt.program",tableno);
    when('5')
      call display("drinks.program",tableno);
    when('6')
      call display("specsl.program",tableno);
    when('7')
      call display("chkout.program",tableno);
    otherwise;
  end;
end;
return;
```

```
TERM:
return;
```

The SCL program in the sub-menus uses another entry statement to accept the table number value that the calling

program passes to it. The entry statement defines the variables that will hold the passed values. The following code fragment shows how the sub-menu would be passed the table number. The MAIN section would perform the work of storing the table number and items selected in a SAS data set.

```
entry tableno $2;
```

```
MAIN:
```

```
/* code to process and store selections */
return;
```

### CONDITIONAL WINDOWS

Manager Raymond was satisfied with the system but had been approached by the Alcohol and Beverage Commission concerning enforcement of the drinking age. To improve the Cafe's ability to enforce the legal drinking age, the consultants were asked to develop some means of ensuring that the staff verified the customer's age. After studying these additional needs, it was decided that when drink option was chosen, a query window would appear and prompt the waitperson to enter the customer's birth date before continuing. If the customer was not old enough, a drinks menu that does not contain alcoholic drinks would appear. The display is a small window that asks for a birth date or allows the staff to bypass the window by omitting the date of birth if no alcoholic drinks were requested.

Date of Birth: \_\_\_\_\_ (mm/dd/yy)

Because it isn't necessary for this query window to occupy the entire screen, the general attributes screen for this program entry was used to specify the window size and placement. This causes the window to appear "on top" of the main menu when it is displayed. The window position part of the general attributes screen looks like this:

```
Start row: 15      col: 20
Number of rows: 5  cols: 20
```

The field attribute screen for this window shows a SAS format and informat assigned to the date field, so that date to numeric conversion will be automatically performed by the program entry. Note that an alias was not needed in this case because the screen field was defined as &dob\_\_\_\_\_ on the display screen in Proc Build.

```
Field name: DOB
Type: NUM
Format: mmddyy8.
Informat: mmddyy8.
```

The source code for the query window calculates the age of the customer and displays the appropriate drinks sub-menu:

```
INIT:
```

```
return;
```

```
MAIN:
```

```
if dob ^= . then do;
    age = intck('year',dob,today());
    if age >= 21 then
        call display('alcohol.program',tableno);
    else call display('nonal.program',tableno);
end;
else call display('nonal.program',tableno);
return;
```

```
TERM:
```

```
return;
```

### EXTENDED TABLES

Because the house specials change daily, the Cafe needed a way to easily change the items listed on the specials sub-menu. If a basic program entry was used for this menu, the display screen, field attribute screens, and the SCL program would all have to be modified every time the specials changed. As there was no programmer on staff at the Cafe, the consultants decided it would be best to use a data-driven extended table for this sub-menu.

"Data-driven" means that the descriptions and prices of the specials would be stored in a SAS data set, and the data set would be used to create the items in the extended table menu. With this method, changing the daily specials would be as simple as updating a SAS data set in SAS/FSP.

The consultants stored a sample of specials in a data set called specials.data, which contains three variables: an abbreviated item name, an item description, and a price. The display screen for the program entry containing the extended table looks like this in Proc Build:

| House Specials |        |
|----------------|--------|
| ***            |        |
| &desc_____     | &price |

The source code for the extended table program entry retrieves the data from the specials data set and populates the extended table with the descriptions and prices of each item. The setrow call defines the number of items in the list, the number of choices allowed, and the selection action. In this example, a dynamic extended table is

created by setting the number of rows to zero (the first parameter) and 'Y' as the last parameter in the call setrow. This allows the extended table to create as many rows in the table as there are observations in the data set at the time that the program is run. The second and third arguments in the call setrow indicate that only one selection should be allowed and the selection window should be closed after the selection is made.

```
INIT:
  /* open data set in read-only mode */
  dsid = open('specials.data','i');
  call set(dsid);
  /* define the extended table */
  call setrow(0,1,'A','Y');
  return;

MAIN:
  /* process and store choice selection */
  return;

TERM:
  rc = close(dsid);
  return;

GETROW:
  /* fill the table with data */
  if fetchobs(dsid,_currow_) ^= 0
    then call endtable();
  return;

PUTROW:
  /* select specific observation */
  rc = fetchobs(dsid,_currow_);
  return;
```

The general attributes screen for this entry was set to EXTENDED TABLE in the "System options" field so SAS/AF knows that this program entry contains an extended table. The field attributes screen also specified YES for the field protection so that the user could not type in the selection list area. When executed, the extended table will appear as shown below:

| House Specials     |         |
|--------------------|---------|
| Beef Brochettes    | \$10.95 |
| Pasta Primavera    | \$ 8.50 |
| Chicken Scallopini | \$ 9.95 |
| Seafood Florentine | \$12.95 |
| Veal Chops         | \$11.00 |
| Blackened Catfish  | \$ 9.50 |
| Leg of Lamb        | \$13.50 |

## BLOCK MENUS

One of the drawbacks of the menu system was that it required the staff to type letters or numbers using a keyboard in order to make selections on most of the

menus. Some staff did not know how to type and were having trouble finding the letters on the keyboard. The maintenance of the menu was also a problem because if new categories were added, deleted, or changed, the display screen, field attributes screen, and the source program all had to be changed. In this respect, the system wasn't flexible enough to meet the changing needs of the restaurant.

Although basic program entry menus can be set up so that no typing is required, to overcome both of these problems, the AF program entry was changed to take advantage of the SCL block function. The block function provides a way to create a user friendly menu display where no data entry is required. To make a selection, the user only needs to position the cursor on the desired selection and press the ENTER key or click the mouse button. The point-and-click feature of a block menu minimizes typographical errors and makes the menu seem easier to use.

Block menus also make menu maintenance easier because the menu is created and controlled entirely within the Screen Control Language program. The display screen and the field attribute screens are not used at all.

The consultants created a new AF program entry, entered the basic framework for the SCL program (empty INIT, MAIN, and TERM sections), and then in the MAIN section, typed the block function.

The block function requires a minimum of 15 arguments and returns a number corresponding to the user's selected choice. The syntax of the block function is as follows.

```
choice=block(window-name,title,color-scheme,
             text-1,text-2,...,text-12);
```

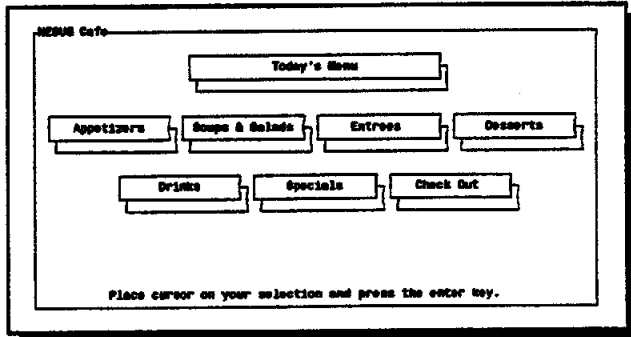
The first argument supplies the name of the window that will appear in the window border. The second argument is the title of the menu that will appear on top of the menu selections when the menu is displayed. The third argument is a number between 0 and 31 that corresponds to the desired color scheme. (For black and white monitors, specify 0 for the color scheme.) The next 12 arguments are the text that will be displayed in each selection box on the menu.

Although all 15 arguments in a block function must be specified, a blank character value can be used for any argument that isn't wanted, except the title and color scheme arguments. Selection text arguments that are blank do not create empty blocks on the menu; the blocks simply do not appear on the menu.

Since the restaurant menu only needed seven selection blocks, the block function that was used looked like this:

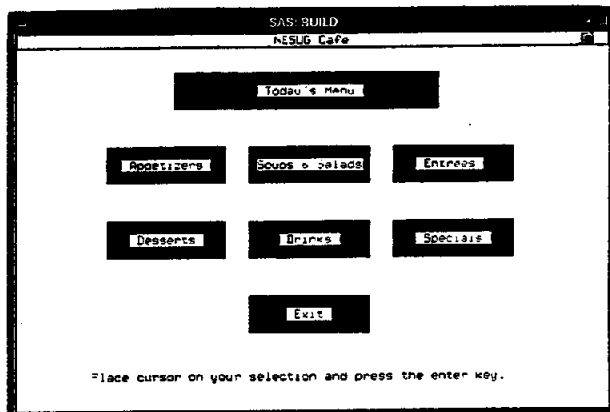
```
choice=block('NESUG Cafe', "Today's Menu", 6,
    'Appetizers', 'Soups & Salads',
    'Entrees', 'Desserts', 'Drinks',
    'Specials', 'Check Out', '', '', '',
    '', '');
```

This block function will produce a menu that looks like this when displayed from an ASCII type terminal:



Notice that this block function example produced a menu with four items on the first row and three items on the second row. The block function creates selection blocks using each text selection argument starting in the first row on the left, and continuing to the right until a maximum of four selections are on the row. To change the way this looks, they rearranged where the blank text selection arguments appeared in the block function. To change the menu so that three items were on the first row, three on the second, and the check out selection by itself on the third, the block function was changed to this:

```
choice=block('NESUG Cafe', "Today's Menu", 6,
    'Appetizers', 'Soups & Salads',
    'Entrees', '', 'Desserts',
    'Drinks', 'Specials', '',
    'Check Out', '', '', '');
```



Whatever selection the user makes is returned by the block function as a selection number (1-12) and stored in the SCL variable "choice." The value of this variable in the SCL program determines which AF program entry should be branched to for display of the next menu. A select function

provides the easiest way to test this variable.

```
select(choice);
  when('1')
    call display("appetz.program",tableno);
  when('2')
    call display("soused.program",tableno);
  when('3')
    call display("entree.program",tableno);
  when('4')
    call display("dessrt.program",tableno);
  when('5')
    call display("drinks.program",tableno);
  when('6')
    call display("specls.program",tableno);
  when('7')
    call display("chkout.program",tableno);
  otherwise;
end;
```

When the block menu is displayed, the cursor is placed on the first selection by default. This action can be overridden by assigning a value to the choice variable before the block function in the SCL program. Adding the statement `choice = 5;` before the block function will cause the cursor to be positioned on the drinks selection when the menu is first displayed.

In addition to executing selections on the block menu, the value returned in the choice variable can be used to branch to help windows. If the user positions the cursor on the entree selection and presses the help key, a value of -3 (the inverse of the selection value) will be returned to the choice variable. To handle the help option, simply add a -3 clause to the selection function.

```
when(-3) call display("entree.help");
```

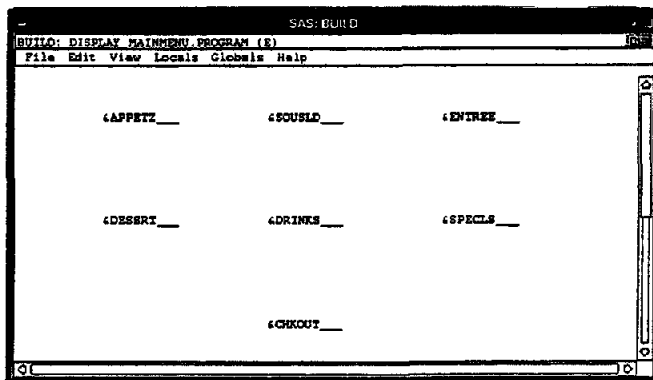
## ICONS ON BASIC PROGRAM ENTRIES

Beginning with SAS Version 6.07, SAS/AF was supplied with a set of icon "pictures" for use with graphical user interfaces. The number and variety of icons may vary between different versions of SAS, and more will be added in the future. Icons supplied by SAS Institute are numbered 1 to 1,024, but less than 150 different icons were shipped with current SAS releases (See SAS Technical Report P-216, page 293-302 for a list of icons.)

For versions of SAS/AF that support them, icons can be added to program entries very easily. They are typically used to replace push-button fields or boxes on menus. A graphical terminal is needed in order to display the icons. Within an environment that has a mix of graphical and ASCII type terminals, separate program entries are not needed for each terminal type. If icons have been defined in a program entry, and SAS detects that the terminal you are accessing the application with does not support

graphics, SAS automatically substitutes boxes or fields instead of the icons that were defined. This feature greatly simplifies the application development process.

The Cafe was pleased that their menu application was working as the consultants had promised, but Chef Hal asked if the application could be made more "high-tech" looking. It was difficult for the consultants to find a truly appropriate icon for each Cafe menu item, so large numerical symbols with textual descriptions as the lower portion of the icon were used. The consultants first added the icons to the basic program entry menus. The Proc Build display screen that was used looked like this:



Several changes were made to the field attribute windows. Each menu selection item was assigned an icon number in the "List" portion of the attribute screen. This number must be surrounded with slashes (/). Each field type was also changed to PUSHBTNC, which indicates that the field will be used like a push button. The text in the "Initial" part of the attribute screens defines the text that will appear under the icon when it is displayed. These were the items in the field attribute screens that were changed:

```
Field name: APPETZ
Type: PUSHBTNC          Protect: INITIAL
List: /141/
Initial: Appetizers
```

The source code was changed to take advantage of the new SCL option which executes individual labeled sections of code when a user modifies or selects the field named in the label. The source code fragment below shows the added statements.

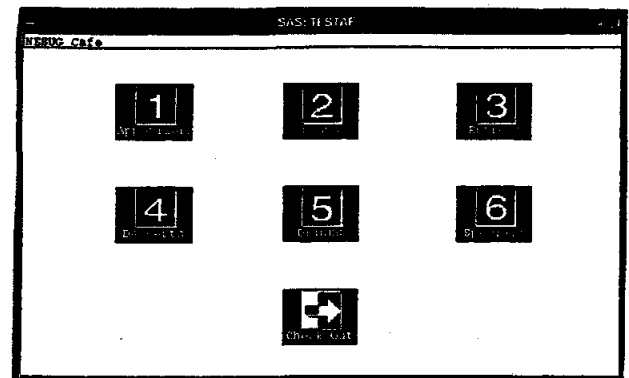
```
INIT:
  control label;
return;

APPETZ:
  call display('appetz.program',tableno);
return;
```

```
SOUSLD:
  call display('sousld.program',tableno);
return;
```

```
[ . . . ]
```

The INIT section contains the control label statement, which causes blocks of code labeled with window field names to execute before execution of the MAIN section when the display field is modified or selected. Therefore if Soups & Salads is chosen by pressing enter or clicking the mouse button, the call display statement is executed immediately. When executed from a terminal that supports graphics, the display screen appears to the user as:



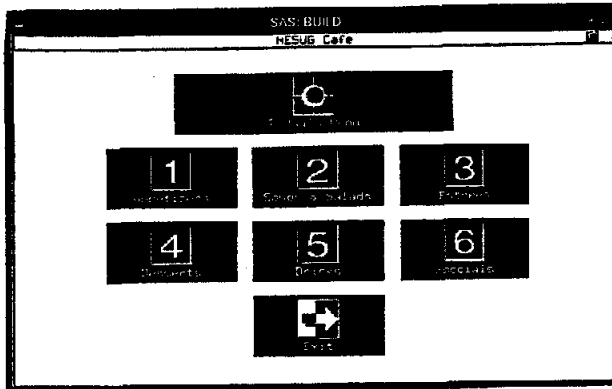
## ICONS ON BLOCK MENUS

For ease of maintenance, the consultants decided it would be better to add the icons to the block menus rather than the traditional program entries. The block function creates the display panel for the developer. The normal block function does this by placing the text of each of the selections in a small box on the screen, with a large box at the top of the menu for the menu title. This type of block menu can be displayed on any ASCII terminal. If the terminal supports reverse video, shading, or highlighting, the block function automatically adapts to the terminal type to enhance the appearance of the menu with these features. On true graphical terminals, however, the block function can also display icons instead of boxes.

Adding icons to the block function is just a matter of specifying SAS icon numbers after the selection text arguments in the function. Use icon number 0 to specify that no icon should be displayed for blank fields. Adding icon numbers does not limit the application to being used only on graphical environments. If icons have been assigned in a block function but SAS detects that the application is running on an ASCII type terminal, the block function will automatically substitute boxes for the icons so that the application can be displayed. The consultants added icons to the previous block function, as shown below.

```
choice=block('NESUG Cafe', 'Today's Menu', 6,  
            'Appetizers', 'Soups & Salads',  
            'Entrees', '', 'Desserts', 'Drinks',  
            'Specials', '', 'Check Out', '', '',  
            '', 141, 142, 143, 0, 144, 145, 146,  
            0, 111, 0, 0, 0);
```

For users with graphical terminals, the display screen will appear as shown below.



### FUTURE PLANS

Although icons were a good idea for this customer, icon pictures that would be appropriate for a restaurant could not be found in the current versions of SAS. Most of the icons supplied with SAS have computer industry or statistics related pictures. Some versions of SAS have a feature that allows the user to create or import an icon. Because SAS reserves icon numbers 1 to 1,024, the first user-supplied icon would be numbered 1,025. The user-supplied icons feature currently works with, or is planned for, these versions of SAS: 6.06 on OS/2, 6.08 under MS-Windows and OS/2, 6.09 or 6.10 under Unix. SAS Institute will be adding support for user-supplied icons for all platforms in subsequent releases.

Also planned for release in the near future are versions of SAS/AF that use object oriented programming technology. The new frame entry types in SAS/AF promise to provide an easy modular way of building menus and menu systems. The soon to be released SAS/EIS product uses object oriented technology with easy to use menus that allow the applications developer to rapidly prototype systems and build complex applications.

### REFERENCES

*SAS/AF Software: Usage and Reference, Version 6, First Edition*, SAS Institute, Inc., Cary, NC

*SAS Screen Control Language: Reference, Version 6, First Edition*, SAS Institute, Inc., Cary, NC

*SAS Screen Control Language: Usage, Version 6, First Edition*, SAS Institute, Inc., Cary, NC

*SAS Technical Report P-216, SAS/AF Software, SAS/FSP Software, and SAS Screen Control Language: Changes and Enhancements, Release 6.07*, SAS Institute, Inc., Cary, NC

### AUTHORS

John Paul Elrod  
Johns Hopkins University  
300 Whitehead Hall  
Baltimore, Maryland 21218  
(410) 516-8732  
jpe@jhuspo.ca.jhu.edu

Marge Scerbo  
University of Maryland at Baltimore  
100 N Greene Street, Room 211  
Baltimore, Maryland 21201  
(410) 706-2495  
mscerbo@umab.umd.edu

SAS®, SAS/AF®, and SAS/FSP® are registered trademarks of SAS Institute Inc., Cary, North Carolina