Scheduler Issues in The Transition to Blaise

Leonard Hart, Mathematica Policy Research, Inc., US

Abstract

Mathematica Policy Research (MPR), a UNIX CASES user, has added Blaise 4 Windows as a CATI platform. One of the major issues in this transition is the standard Blaise Scheduler. In this paper I will describe MPR's scheduler needs, where the Blaise scheduler meets those needs, and where work-arounds need to be developed. The paper will more broadly touch on scheduling issues in a multi-platform environment, and the Blaise Inherit CATI mode in general.

Introduction

The Blaise call scheduler or, as referred to in the Blaise Developer's guide, the "CATI call management system," is a powerful tool in controlling the delivery of cases to interviewers. One simple line of code, "Inherit CATI", included in your program, takes this highly complex procedure and converts it into a easy to use parameter driven system. Ease of use comes with a cost however; it limits you to a given format for CATI operations. As we all know, no two surveys are exactly alike and each has its own special needs. This problem is compound by diverse procedures among survey organization. In this paper are a few brief examples on how we at Mathematica Policy Research (MPR) looked outside the box to use the call scheduler to meet our needs. Our goal was not to recreate the Blaise CATI management process but to enhance it.

Background

As Mathematica Policy Research looked into converting to Blaise, one of the deciding factors was the CATI Call scheduler process built into the system. As we started to gain experience with the Blaise family of applications, we started to note the limitations of the CATI call management system. At this point we sat down as a group at MPR and tried to decide how can we overcome these obstacles. Basically we had two choices; write our own CATI call management system or try to build on the Blaise CATI call management system.

Building our own system would yield a CATI management process that could allow us to customize the CATI management system as needed to meet specific survey requirements. The second choice was to use the Blaise CATI call management system and see if we could enhance it to meet each survey's requirements. We opted to go with the second choice once we started to look at the complexity of trying to develop and maintain our own CATI call management system. One of the key factors in our decision was based on our past experience with our other CAI system. We also felt that the introduction of the Blaise Component Pack, along with Manipula, Maniplus and Delphi, could help us meet our goal.

Call History

One of the first major challenges we encountered using the call scheduler was the need to maintain a complete call history of each case, along with other key information that we felt necessary. The current Blaise CATI call management system only keeps a call record of five attempts; the very first contact and the last four attempt. The call history between these attempts is lost. We needed the capability to keep all

attempts. This information is used in creating productivity reports, management of the survey and on help in addressing problems as the survey progresses. Figure 1 depicts the scheme currently kept by the call scheduler.

Figure 1.

```
Block TCall
        WhoMade    "who made last dial in call     ": STRING[10]
        DayNumber  "daynumber relative to FirstDay ": 1..999
        DialTime   "time of last dial in call      ": TimeType
        NrOfDials  "number of dials in call        ": 0..9
        DialResult "result of last dial in call    ": TcallResult
Endblock
```

An additional problem with the current Blaise CATI call management history concerns the type of data being kept. We wanted to keep additional information such as end time of the call, current case status and interviewer notes to help us manage the survey process and to track interviewer performance. Our first thought was to modify the CATI.INC file by increasing the size of the array for Regscalls (see figure 2) so that it would keep track of all calls. This is the block of code the Blaise CATI management system uses to store information. Although increasing the array size for Regscalls worked, we had some concerns. First, what effect would modifying CATI.INC have on the Blaise system? The Blaise development team felt that this would not create a problem with any of the other Blaise applications. Second, as we upgrade to a newer version of the Blaise system it would require us to make changes to CATI.INC again. It would then be necessary to verify that these changes would not affect the other Blaise applications.

Figure 2.

```
FIELDS
    NrOfCall “number of calls” : 1.99
    FirstDay “date first call” : DATETYPE
    RegsCalls “registered calls” : ARRAY[1..5] OF TCALL
```

The next question was how to get the additional items of information we want to retain included in the CATI.INC file. This was a harder task. Our first attempts to save this additional information to the modified CATI.INC resulted in failure to write the information to this block at the proper time. We also needed to ensure that we didn’t re-update our additional call information as an interview progressed. And once again, going this route meant that more changes would have to be made with each new version of Blaise. The more we worked on this solution, the less satisfactory it seemed to our problem. After some research we came across a prepared for presentation but not presented at the May, 2000 Blaise user conference in Ireland by Linda L. Anderson, Statistical Laboratory at Iowa State University.

Anderson's paper, "Displaying a Complete Call History to Interviewers", addressed the same problems we were facing; how to keep a complete call history and displaying the information for the interviewers to use. In her paper she looked at two different approaches to accomplishing these goals. The first approach was adding the key information to the history file. The disadvantage of this according to Anderson, is that the necessary information would not be available in a timely fashion. A process would have to be run against the history file to populate the information into the external Blaise dataset. For this process to work efficiently and avoid corruption of the external data file, all users and applications accessing the dataset would have to be suspended for the duration of the effort. Anderson's second approach involves the construction of a block of code to be included within the Blaise dataset to maintain the additional information. We choose this basic approach and expanded upon it.

In Anderson's approach, information is displayed once the case is loaded, after the dial screen. We wanted to display this information on the dial screen before the call was made. This format would allow interviewers a chance to review the complete call history with any interviewer notes before initiating the call directly on the dial screen (see appendix A for example of the screen). We achieved this by condensing and reformatting our own history block (BCall) to a field called CallHistNotes (see appendix B for example of the Blaise code). In this field we include who made the call, date of the call, the dial time, current status and the first 50 characters of the note field. We condense the note field to 50 characters to make the screen more user-friendly, since our note fields are defined as a STRING[32000]. On the dial screen after the CallHistNotes field, we also include the unedited note fields, allowing the interviewer to read the full notes if they like. The interviewer would normally have to scroll down to see this information.

By creating our own CATI call history block, we successfully overcame our problems with the default Blaise CATI call history block; however, we were getting multiple entrees for each session in the start. We wanted to move the call history information down one level in the array after leaving the dial screen so the latest information is always the first position of the array. We accomplished this by using the insert(1) to our array, a solution that was initially problematic. Every time information changed in the CallHist block everything was moved down one level, resulting in multiple increases in our CallHist block. We also discovered that merely loading the case to the dial screen also caused the information to move down one level. Further more, when the case was brought up in conduct interview mode, the information in the array was shifted down once again. We overcame both of these problems by setting an AUXFIELD TempID to 1 (see figure 3) after the first Insert(1) was done. We used the field TempID as an indicator to tell us if the insert command had been incremented. Since the TempID field is not on the route, we had to use the KEEP method to maintain its value.

Figure 3.

```
TempID.KEEP
IF TempID = EMPTY "" THEN
  CaseHist.INSERT(1)
  TempID := 1
ENDIF
```

These modifications to our own call history block have basically met our basic needs although some refinements are needed to the process. The interviewers would like to see more information on the dial screen, but the current layout of the dial screen limits the amount of information we can display. Therefore, our next goal is to write a popup screen that would allow for us to display more information on

the screen in a user-friendly layout before the interviewer makes the call. With the introduction of the Blaise Component Pack, we feel we can write a Visual Basic program to clearly display this information.

Complex includes/excludes in daybatch

One of the options when creating a daybatch is to include or exclude cases based on a field in the Blaise dataset using the CATI Specification program. This works well and would be easy to implement if inclusion and exclusion needs were uniform across instances. Unfortunately, in some cases, complex exclusion or inclusion rules are necessary. For example, cases may be excluded based on a formula. The current CATI specification program does not allow such calculations.

We overcame this problem by creating a field in the Blaise dataset called CaseOnOff with a value of zero or one. Before we run the CATI Management program to create the daybatch, we first run a Manipula program to set the value of CaseOnOff to a zero or one, based on multiple criteria. The Manipula code listed in appendix C is an example of how we reactivate a case after a given number of days based on several fields. In this example we first look at two different fields to see if they meet a certain criteria; if so we then calculate how long this case has been on held status. If it has been last then fourteen days we turn the case off by setting the CaseOnOff field. Then in the CATI specification program we can include or exclude cases based on the CaseOnOff field.

Auto Dialer

When our interviewers make a call, a billing code must be entered before they get a dial tone. Each survey has its own billing code. Since our interviewers use a modem to make the call, we needed a way to pass along the billing information through the modem to the PBX system. Several options were available to us. One such option was to attach the billing code to a phone number prior to populating the sample information into the system. Another recourse involved direct modification of the dialer options.

Attaching the billing code to the phone number raised several problems. For example, if the interviewer needs to change the phone number, it would risk a corresponding change in the billing code when making the phone number changes. What if the interviewer was entering a brand new phone number would they remember to include the billing code; if so, would it be the right one. The second problem was if you had multiple phone numbers you would need to change the billing code for each of them. Our conclusion by adding the billing code to the phone number would increase the chance for errors.

The second option of setting the dialer options imposed its own problems. Here the problem is each machine might be used for different surveys, each having a different billing code. The question became how to set the billing code for each survey. This process was further complicated by our setup of Windows 2000, which limits users capabilities to implement modifications, thus making it more difficult for programmers to accomplish changes behind the scenes.

Our first approach to tackling this problem was including code in our Visual Basic menu system that would pass billing information for the dialer to the registry. As mentioned previously, the interviewer rights are limited on the machine and one of these limitations is the ability to update the registry.

The next approach involved the use of a macro scripting language called Winbatch. Winbatch allowed us the ability to script the needed changes to the dialer options. Our WinDial (see appendix D) script is generic enough to allow us to pass parameters through to it by our Visual Basic interface, thereby allowing each survey to set the proper PBX codes for the dialer. This process is done by executing WinDail twice through our Visual Basic menu system. During the first execution of WinDial the Visual Basic menu calls WinDail and sets the proper dialer options for a given survey. The second time, it hides

the dialer from the taskbar. Under normal Blaise calling circumstances, the dialer is minimized to the taskbar, but we found that this was confusing to our interviewers. We could have hide the taskbar but that would have lead to other problems. Hiding the dialer from the taskbar eliminated this problem.

Conclusion

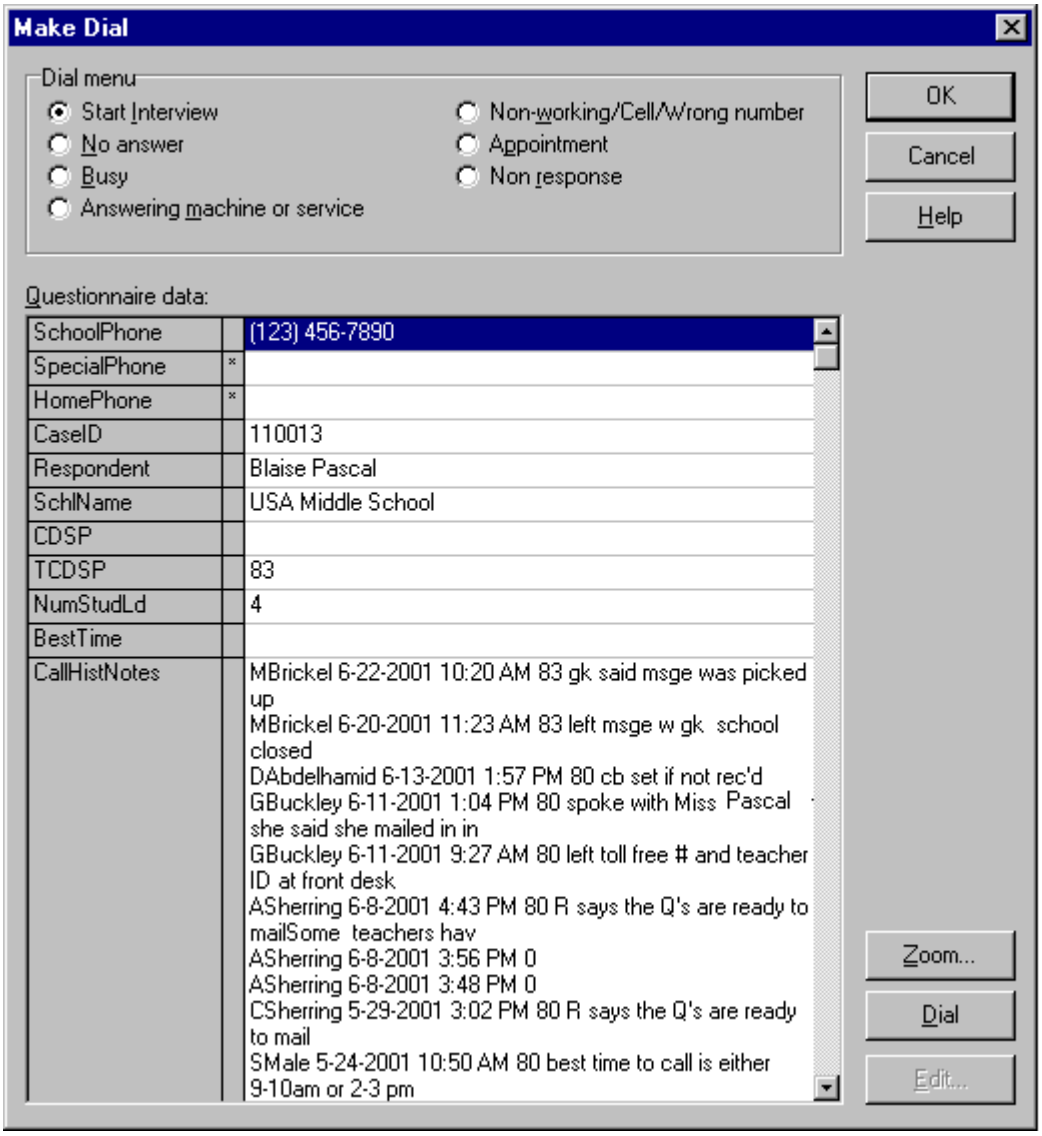
As the Blaise community grows, more demands will be placed on the Blaise development team to expand the capabilities of the Blaise CATI call management system. Like any development team they have a limited amount of resources to apply to this area. I urge us as a community to work together to compile a list of needs that we feel that should be included in the scheduler and then present these items to the Blaise development team. Past experience with the Blaise development team has been that if modifications are desired by the community, most likely they can be built into the system. If not a reason will be given for its lack of feasibility.

In the meantime, I urge you to think outside the box to make the CATI management system work for you. You will be surprised what you can accomplish with the Blaise family of applications. In addition, with the recent release of the Blaise Component Pack and other programming and scripting languages, you should be able to do about anything you want to enhance your CATI management system without rewriting the Blaise CATI call management system. We all like turnkey applications, but as mentioned previously no two surveys are alike. Hopefully, through these examples and by working together as a community, we can make the Blaise CATI call management system fit anyone's needs.

References

Anderson, L (2000). Display a Complete Call History to Interviewers. Proceedings of the Sixth International Blaise users conference, Kinsale, Ireland, CSO Ireland.

Appendix A


Make Dial
Dial menu
Start Interview
No answer
Busy
Answering machine or service
Non-working/Cell/Wrong number
Appointment
Non response
OK
Cancel
Help
Questionnaire data:
SchoolPhone
(123) 456-7890
SpecialPhone
HomePhone
CaseID
110013
Respondent
Blaise Pascal
SchlName
USA Middle School
CDSP
TCDSP
83
NumStudLd
4
BestTime
CallHistNotes
MBrickel 6-22-2001 10:20 AM 83 gk said msge was picked up
MBrickel 6-20-2001 11:23 AM 83 left msge w gk school closed
DAbdelhamid 6-13-2001 1:57 PM 80 cb set if not rec'd
GBuckley 6-11-2001 1:04 PM 80 spoke with Miss Pascal she said she mailed in in
GBuckley 6-11-2001 9:27 AM 80 left toll free # and teacher ID at front desk
ASherring 6-8-2001 4:43 PM 80 R says the Q's are ready to mailSome teachers hav
ASherring 6-8-2001 3:56 PM 0
ASherring 6-8-2001 3:48 PM 0
CSherring 5-29-2001 3:02 PM 80 R says the Q's are ready to mail
SMale 5-24-2001 10:50 AM 80 best time to call is either 9-10am or 2-3 pm
Zoom...
Dial
Edit...

Appendix B

```
BLOCK BCallHist
 BLOCK BCall
  FIELDS
   WhoMade : STRING[20]
   DailTime : TIMETYPE
   EndTime : TIMETYPE
   DailDate : DATETYPE
   IntNotes : STRING
   TCDSP : 1..999

  RULES
   WhoMade.KEEP
   DailTime.KEEP
   DailDate.KEEP
   EndTime.KEEP
   IntNotes.KEEP
   TCDSP.KEEP
 ENDBLOCK

 FIELDS
  CaseHist :  ARRAY[1..30] of BCall
  CallHistNotes : STRING[32760]

 AUXFIELDS
  TempID, I,J : INTEGER

 LOCALS
  K : INTEGER

 RULES
  I := 1
  j.KEEP
  j := 1
  FOR K := 1 to 30 DO
   CaseHist[K].KEEP
   IF CaseHist[k].WhoMade = RESPONSE "" THEN
            CallHistNotes := CallHistNotes + CaseHist[k].WhoMade + ' ' +
            DatetoStr(CaseHist[k].DailDate) + ' '  + TimetoStr(CaseHist[k].DailTime) + ' ' +
            STR(CaseHist[k].TCDSP) + ' ' + SubString(CaseHist[k].IntNotes,1,50) + '@/'
   ENDIF
  ENDDO
  If I = J "" THEN
   TempID.KEEP
   IF TempID = EMPTY "" THEN
    CaseHist.INSERT(1)
    TempID := 1
   ENDIF
   CaseHist[1].WhoMade := StafInfo.User
   CaseHist[1].DailTime := StrtTime
```

```
      CaseHist[1].DailDate := sysdate
      J := J + 1
    ENDIF
ENDBLOCK

FIELDS
  CallHist : bCallHist
```

Appendix C

```
AUXFIELDS (GLOBAL)
  CurrentJulian : INTEGER

PROLOGUE
  CurrentJulian := Julian(SYSDATE)

MANIPULATE
  IF (Mangmnt.CDSP = EMPTY) THEN
    IF (CallHist.CaseHist[1].TCDSP = 21) THEN
      IF Julian(CallHist.CaseHist[1].DailDate) + 13 > CurrentJulian THEN
        Mangmnt.CaseOnOff  := 1
      ELSE
        Mangmnt.CaseOnOff  := 0
      ENDIF
    ENDIF
  ENDIF
```

Appendix D

```
; A Windows dialer, based on HelpDial.wbt
IntControl(50,0,0,0,0) ; Turn off Web Page Support
IF Param0 != 1 && Param0 != 3
      CR = StrCat(num2char(13),num2char(10))
      msg = WinExeName("")
      msg = StrCat(msg," - Version: ",FileTimeGet(msg))
      txt = "Called with 1 parameter (/I or a phone #), this works as the
            Watcher/Dialer.%CR%With 3 parameters, it acts as SetDial%CR%%CR%SetDial Options:     LocalAreaCode
            Message(msg,StrCat(txt,"Note that these options are all upper case and are mutually
            exclusive%CR%%CR%If the first arg is a phone # and contains spaces, be sure to quote it!"))
      Exit
ENDIF
IF WinExist("Phone Dialer") == @FALSE
IF WinState("") == @HIDDEN THEN
      RunHide("Dialer","")
ELSE Run("Dialer","")
      WinWaitExist("Phone Dialer",-1)
ENDIF
IF Param0 == 3
      SendKeysTo("Phone Dialer","!td!e!c%Param1%{TAB}%Param2%{TAB}%Param3%~~")
      WinClose("Phone Dialer")
      msg = "Dialer Parameters have been set..."
      goto theEnd
ENDIF
IF "%Param1%" != "/I"
   ClipPut(Param1)
   SendKeysTo("Phone Dialer","!n{Del}+{Ins}~")
ENDIF
IF WinExist("Yet Another Windows Dialer")
   msg = "Another Watcher is already running..."
ELSE
   WinTitle("","Yet Another Windows Dialer")
   WHILE WinExist("Phone Dialer") == @TRUE
         WinWaitExist("Active call",1)
         IF WinItemizeEx("Active call",0,0) != "" THEN SendKeysTo("Active call","~")
         IF WinItemizeEx("Warning",0,0) != "" THEN SendKeysTo("Warning","~")
   ENDWHILE
   msg = "Goodbye!"
ENDIF
:theEnd
tme=TimeDate()
Display(5,"WinDial exiting at %TME%...",msg)
```