Programming Blaise for a Multi Questionnaire Study

Youhong Liu, Heidi Guyer
Survey Research Center
the University of Michigan

1. Introduction

The Panel Study of Income Dynamics (PSID) is a longitudinal survey of a representative sample of U.S. men, women, children and the families in which they reside. Data on employment, income, wealth, housing, food expenditures, transfer income, and marital and fertility behavior have been collected annually or bi-annually since 1968. From 5,000 families in 1968, the study has grown to include over 10,000 families. In all interactions with our study members, we use the name the Family Economics Study (FES).

The Child Development Supplement (CDS) was initially fielded in 1997; the U.S. had very limited nationally representative longitudinal studies of children focusing on child health and development in the preadolescent years. CDS was designed to increase our knowledge about how these and other changes are influencing child development and well-being.

The original CDS followed a cohort of children in PSID families who were 0–12 years of age in 1997 through three waves of data collection and focused on understanding the socio-demographic, psychological, and economic aspects of childhood in an on-going nationally representative longitudinal study of families. In 2014, all of the children in the original cohort have reached adulthood, and a new generation of children has replaced them in PSID families. The goal is to collect information in 2014 on all children aged 0–17 years in this new generation, shifting the orientation from a cohort study to one that obtains information on the childhood experiences of all children in PSID families. The sample will consist of approximately 6,400 children aged 0-17 and 3,500 primary caregivers.

In this paper, we will discuss the Blaise implementation for the most current wave of CDS, conducted in 2014.

2. CDS 2014 Respondents and Blaise Instruments

There are three kinds of respondents: COVERSCREEN, THE CHILD and PRIMARY CAREGIVER (PCG).

With these three kinds of respondents, the Study is defined with three Blaise Instruments. In the chapters below, we will discuss the Blaise programming techniques used for each of these instruments.

3. Coverscreen Instrument

The Coverscreen is used to verify information about the preloaded CDS children and other family member to identify who each child's primary caregiver (PCG) is The program then generates the appropriate number of sample lines for each respondent.

The Coverscreen is the instrument preloaded for each household in the sample. Since it is used to obtain information that determines the respondents as well as the structure and the flow of all the other portions of the interview, it must be completed before any of the PCG or Child sample lines are spawned. Once the Coverscreen is completed, it cannot be revised or edited without intervention from the Ann Arbor office. This is why it is extremely important that it is programmed accurately. If data are entered incorrectly in the Coverscreen, the consequences can be costly:

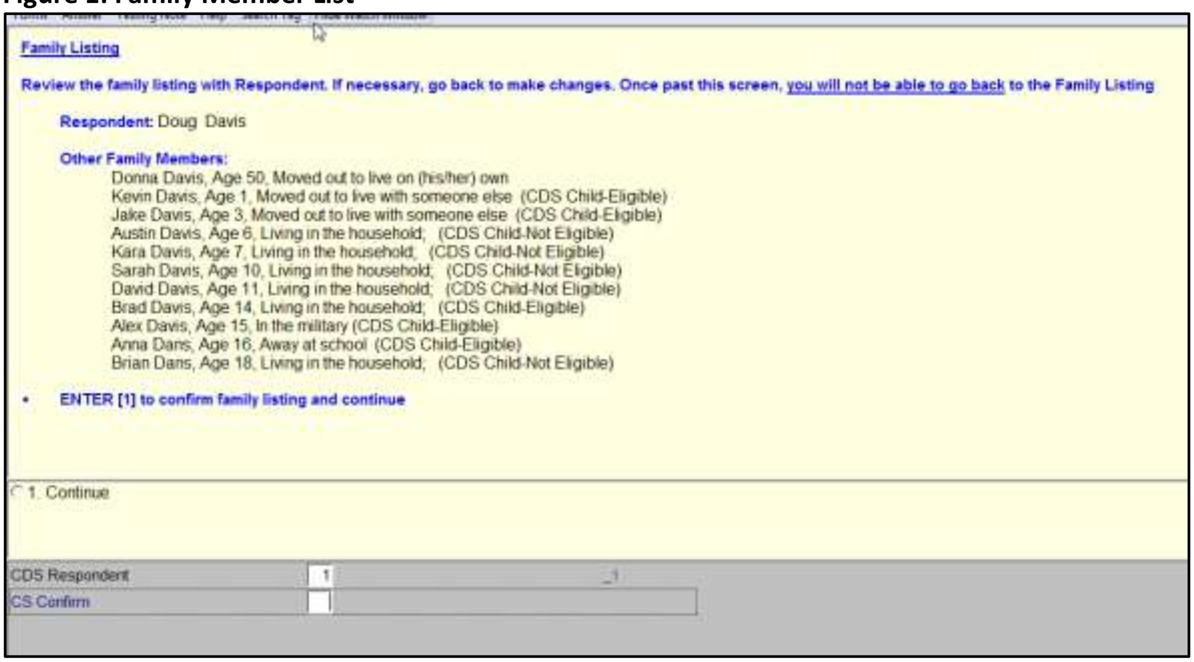
a. Interviewing Family Units (FU) where the CDS child is actually ineligible
b. Not interviewing a FU where the CDS child is eligible
c. Interviewing the wrong PCG
d. Passing bad data to subsequent interviews
e. Incorrect age and grade of the CDS child impacts many skip patterns in all of the interviews.

In order to collect the information in the Coverscreen effectively and robustly, project staff designed several challenging features for Blaise programmers to implement. A few examples are provided in the following sections.

3.1 Children's relationships to others in the household

The Coverscreen instrument starts with the household members table which verifies members names, birthdays and FU status. The result can be complicated in households with many children and/or family members, as shown below:

Figure 1: Family Member List



Following the household member table is the relationship table. The purpose of this table is to identify the CDS children's relationship to other family members, aged 16 or older, who can potentially be the PCG respondent of a CDS child. The table is loaded with all CDS children as columns and potential PCGs in the family as rows. If a CDS child is 16 or older, then he/she is also be selected a PCG. In order for users to navigate the Blaise DEP easily and complete the table quickly, here are some of the requirements from the research staff:

a. In the same row, if a later child's relationship to a FU member is the same as a previous child, then the relationship question will be skipped and a relationship is auto assigned.
b. In different rows, if a child is a sister, brother or cousin of a previous child, then the relationship question is skipped, and relationship is auto converted and assigned.
c. Children list in some of the questions is generated dynamically.

To meet the above requirements, complex programming techniques are implemented. As many as four For/Do loops are used in the section. Keeps on fields and blocks are used so data can be passed properly from parent loops to children loops.

Figure 2: Relationship Table – Code 17 (Sister) is Detected and Converted to Code 13 (Brother)

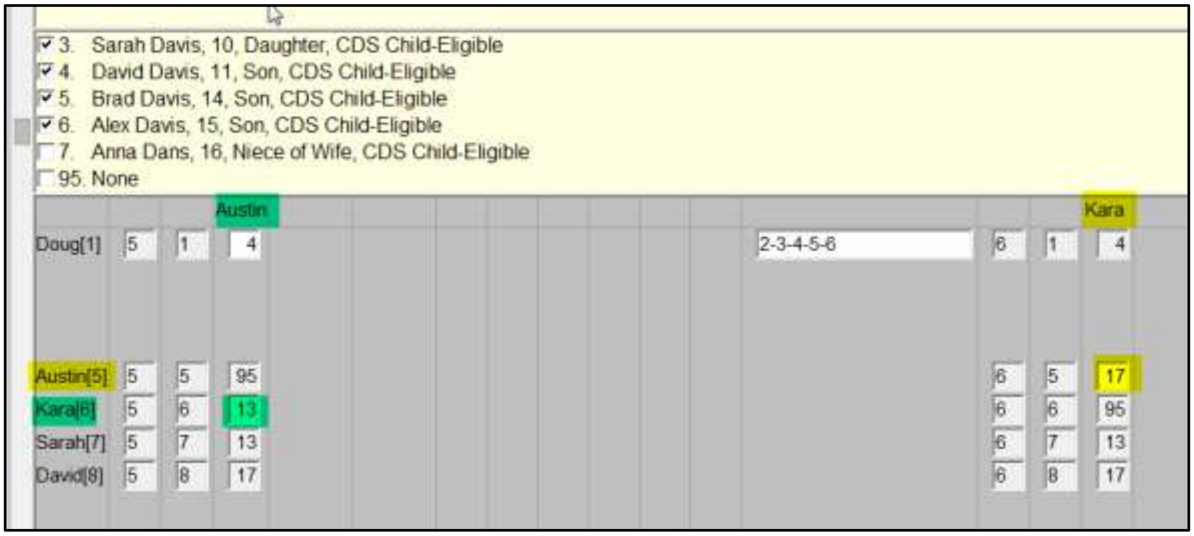


Figure 3: Code Example – Convert Relationship between Sisters, Brothers and Cousins

```
xRTCPrev := EMPTY      {Start empty}
FOR I:=1 TO 24 Do
     IF I<pFmIndex THEN    { Loop Previous Rows}
          FOR J:=1 TO Num_CDSKid DO     {Loop all columns}
               IF RTCFmLstB[I].RTCCDSLstB[J].RTCCDSAQSN =  pRTCFMAQSN      {AQSNs are FU member's IDs}
                 AND  RTCFmLstB[I].RTCFMAQSN = RTCCDSAQSN THEN
                   IF RTCFmLstB[I].RTCCDSLstB[J].RTC >=BSister  AND  RTCFmLstB[I].RTCCDSLstB[J].RTC<=Cousin THEN
                           {Biological/Step/Half/Adoptive Sister/Brother or Cousin }
                           xRTCPrev :=  RTCFmLstB[I].RTCCDSLstB[J].RTC
                           xGender1 :=  RTCFmLstB[I].RTCCDSLstB[J].RTCCDSGender
                           xGender2 :=  RTCFmLstB[I].RTCFMGender
                   ENDIF
               ENDIF
          ENDDO
     ENDIF
ENDDO
IF xRTCPrev<>EMPTY THEN       {Found a match}
   {Procedure Assigning code to the current cell based on previouse Gender and Relationship}
   AssignSibRelCode (xRTCPrev,  xGender1, xGender2, RTC)
   RTC.SHOW
```

3.2 Preload Spawning

As previously described, accurate completion of the Covercreen is very important. It is a driver of the two other Blaise instruments: PCG and Child. These two instruments have very complex preload data structures themselves. Data is pulled from different parts of the Coverscreenand used for fills and logic in the PCG and Child instruments.

There are over 100 preload variables for the Child instrument and 600 preload variables for the PCG instrument. Additionally, the number of Child and PCG lines generated from a Coverscreen can be varies by household. In fact, Coverscreen lines can also be derived from an original Coverscreen, if a child is moved out with or without an adult.

a. Preload Generation

For other studies at the University of Michigan, the sample management system is responsible for pulling out and generating the preload information for spawned lines. Given the complex preload data structures for CDS 2014, it is difficult for a process outside of Blaise to figure out which data goes to which instrument, and what is the logic if a code conversion is needed. It was determined that the best way to generate the preload was to generate the strings inside the datamodel with Blaise code only. The Coverscreen's programmer can perform string creation relatively easily and accurately because of their knowledge of the variable locations and logic.

The actual generation of the preload strings is easy to understand in Blaise. It is done by concatenating variable values, e.g. PreloadStr:=Field1 + Field2. The challenging aspect of generating the preload string is to match and interpret variables between the Coverscreen and the spawned datamodels. With careful design, programming, several rounds of modifications and testing, we found this process was very effective.

Figure 4: Code Example – Child Datamodel Preload Generation

```
Locals
    SCount : INTEGER {for special kid}
RULES
     xTEMP  := PCGKids[pPCGINdex].CDSKids [pKidIndex].KidIndexInFU
     xTEMP1 := PCGKids[pPCGINdex].PCGIndex
     xTEMP2 :=  PCGKids[pPCGINdex].CDSKids [pKidIndex].KidIndex

     PCGPre [pPCGIndex].KidPre[pKidIndex].KidFname :=  CS3Table.HHLListBLOCK[xTEMP].CS3FName
     PCGPre [pPCGIndex].KidPre[pKidIndex].KidLName :=   CS3Table.HHLListBLOCK[xTEMP].CS3LName
     PCGPre [pPCGIndex].KidPre[pKidIndex].KidAQSN  :=   CS3Table.HHLListBLOCK[xTEMP].CS3AQSN
     PCGPre [pPCGIndex].KidPre[pKidIndex].KidPSIDAQSN   :=   CS3Table.HHLListBLOCK[xTEMP].CS3PSIDAQSN
     PCGPre [pPCGIndex].KidPre[pKidIndex].KidCDSAQSN    :=   CS3Table.HHLListBLOCK[xTEMP].CS3CDSAQSN
     PCGPre [pPCGIndex].KidPre[pKidIndex].KidSampleID := SampleID + 'P' + str(pPCGIndex) + 'C' + Str(pKidIndex)
     xTempPre :=''

     xTempPre :=xTempPre + PCGPre [pPCGIndex].KidPre[pKidIndex].KidSampleID + '^' {1}
     xTempPre :=xTempPre +   pXTempS {2-11, passed in from PCG}
     xTempPre :=xTempPre +   str(Elig.ChildEligB[xTemp].ChildElig.ORD)    + '^'   {12,Preload.ChildElig}
     xTempPre :=xTempPre +   CS3Table.HHLListBLOCK[xTEMP].CS3PSIDAQSN  + '^'      {13,Preload.CHPSIDAQSN}
     xTempPre :=xTempPre +   CS3Table.HHLListBLOCK[xTEMP].CS3AQSN    + '^'    {14,Preload.CHAQSN}
     xTempPre :=xTempPre +   CS3Table.HHLListBLOCK[xTEMP].CS3CDSAQSN   + '^'   {15,Preload.CHCDSAQSN}
     xTempPre :=xTempPre +   Preload.PRELOADHHLTable.PRELOAD_HHL [xTEMP].Six8ID + '^'     {16,Preload.CH68ID}
     xTempPre :=xTempPre +   Preload.PRELOADHHLTable.PRELOAD_HHL [xTEMP].PN    + '^'      {17,Preload.CHPN}
     xTempPre :=xTempPre +   str(Preload.PRELOADHHLTable.PRELOAD_HHL [xTEMP].RTH.ORD)   + '^'     {18,Preload.CHRTH}
     xTempPre :=xTempPre +   str(CS3Table.HHLListBLOCK[xTEMP].CS3FU.ORD)  + '^'     {19,Preload.CHFUHU}
     xTempPre :=xTempPre +   CS3Table.HHLListBLOCK[xTEMP].CS3FName  + '^'          {20,Preload.CHFName}
     xTempPre :=xTempPre +   CS3Table.HHLListBLOCK[xTEMP].CS3LName  + '^'          {21,Preload.CHLName}
     xTempPre :=xTempPre +   str(CS3Table.HHLListBLOCK[xTEMP].CS3Gender.ORD)  + '^'            {22,Preload.CHGender}
```

b. Preload Spawning Testing

With the above preload spawning method we were in need of a way to test spawned lines. Our sample management system can perform some integration test, but it is difficult to test different scenarios and combinations within it. During the project development, it became apparent that we needed a feature within our CAI Testing Tool (CTT) that would allow us to spawn lines. This new feature was required for multiple reasons, including:

- Ensure that the number of lines generated is correct
- Ensure the number of preload variables for each datamodel generated is correct
- Confirm values are pulled from the correct Coverscreen fields and are loaded into correct spawn line fields
- Use the generated preload strings to test the Child and PCG instruments, to avoid manual data entry

After reviewing the existing CAI Testing system, we determined that this feature could easily be added.

- In the Coverscreen datamodel, a variable named SpawnData is added, and its question text is a well formatted xml string that can be parsed by the testing system. The xml string consists of preload string fields in the datamodel
- In the testing system, a C# module is developed to exam the xml string and to pull out the value of the fields specified from the Coverscreen and then to generate lines for PCG and Child datamodels This C# module is mainly developed for the CDS study but it can also be used for other studies if the datamodel is programmed accordingly

Figure 5: XML String for Spawn Line

```
AUXFIELDS
SpawnData "<SpawnLine>
            <SpawnDetail cttStudyID = 'CDS14' cttpid =""SRC.SRO.CDS14.CS"" desc=""Split off"">
               <FieldName loop = ""6"">CSBLOCK.PreOutSO.SOPre[x].Str_</FieldName>
            </SpawnDetail>
            <SpawnDetail cttStudyID = 'CDS14' cttpid = ""SRC.SRO.CDS14.Child"" desc=""Child"">
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[1].KidPre[x].Str_</FieldName>
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[2].KidPre[x].Str_</FieldName>
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[3].KidPre[x].Str_</FieldName>
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[4].KidPre[x].Str_</FieldName>
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[5].KidPre[x].Str_</FieldName>
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[6].KidPre[x].Str_</FieldName>
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[7].KidPre[x].Str_</FieldName>
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[8].KidPre[x].Str_</FieldName>
               <FieldName loop = ""15"">CSBLOCK.PreOutPCG.PCGPre[9].KidPre[x].Str_</FieldName>
            </SpawnDetail>
            <SpawnDetail cttStudyID = 'CDS14PCG' cttpid = ""SRC.SRO.CDS14.PCG"" desc=""PCG"">
               <FieldName loop = ""9"">CSBLOCK.PreOutPCG.PCGPre[x].Str_</FieldName>
            </SpawnDetail>
            <SpawnDetail cttStudyID = 'CDS14' cttpid = ""SRC.SRO.CDS14.Child"" desc=""Not Elig"">
               <FieldName loop = ""15"">CSBLOCK.PreOutNotElig.NotEligPre[x].Str_</FieldName>
            </SpawnDetail>
      </SpawnLine>": 1..1 {Xml string for CAI testing system to spawn lines}
```

c. Sample Management System Preload to Spawned Blaise Datamodel

With careful preload generation and testing, the preload task for our sample management system is simplified greatly. A small number of variables are pulled,the values can be saved in text files and then loaded into the spawned lines by a Manipula script as with the other projects.

4. Parallel Block Programming

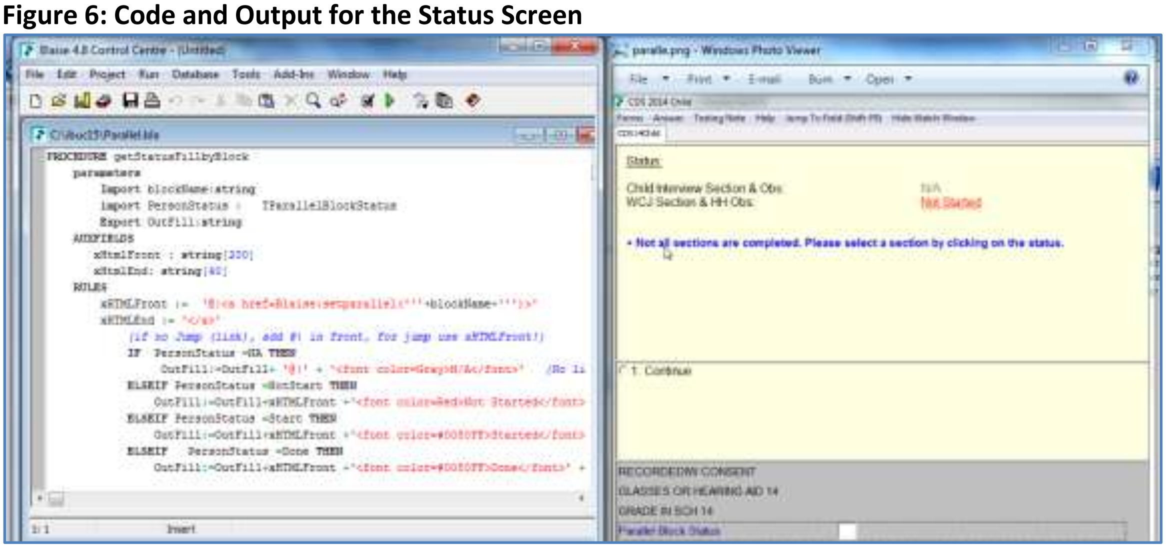
Parallel Block programming enables you to process one or more block fields separately from the current route in the DEP. This technique is widely used in the Blaise community. CDS 2014's Child and PCG datamodels have employed this technique in two ways:

4.1 Access Parallel Blocks on a Status Screen

Based on our previous experience, we were aware that we needed to be extremely cautious about parallel programming method. If it is not programmed properly, wrong statues may be assigned and result in not the main block or some of the parallel blocks not completing as required. To ensure proper programming, we implemented the following:

- All parallel statue generations are made in one procedure instead of in the main body of the datamodel
- Not every block has the same set of statues. We examined and documented the details in the procedure carefully to ensure proper assignments are made
- On the statues screen, startparallel procedure is used to jump into parallel blocks Extreme caution was taken during the programming since some statues can jump, while others cannot jump
- Color codes are used for different statues so it is easy for interviewers to distinguish

Figure 6: Code and Output for the Status Screen



4.2 Parallel Blocks to Enable and Disable the Achievement Tests

There are three achievement tests in the Child and PCG datamodels. It is important that we allow a respondent to stop at any point of a test. To accomplish this, another parallel block feature was implemented inside of a main parallel block (WCJ). There are three sub-parallel blocks used to start and stop the three tests respectively.

These blocks are made visible on the Blaise parallel tabs by adjusting the BXI file. Again, they should be programmed carefully so the tabs are shown for the right tests and data are saved properly for interrupted tests.

Figure 7: Code Example – a Parallel Block for Enable and Disable a Block

```
LWStart
LWask.KEEP   (Grade) {Letter word test questions, keep is used to save both complete/incomplete test data}
LWEND.KEEP
StopBLOCK.KEEP      {Parallel}
IF LWStart<>EMPTY AND LWEND=EMPTY THEN  {Prevent Iver to go back to change data if the test is finished LWEND<>EMPTY}
    IF StopBlock.StopLW  <> Stp  then   {Stop test if the stop button is checked in the parallel block}
        LWAsk     (Grade)     {In the testing block}
    ENDIF
ENDIF
LWEND
```

5. Woodcock Johnson Assessments Programming

The Woodcock Johnson (WCJ) Test of Achievement is a well-known, established educational assessment tool. CDS has three subtests. They are:

- Letter-Word Identification
- Passage Comprehension
- Applied Problems

In the test, the term "basal" is used to refer to when the respondent obtains six or more consecutive items correctly. The term "ceiling" is used when the respondent answers six or more items incorrectly. Blaise programming is required to automatically calculate the basal and ceiling using the codes entered by interviewers and determine what items should be given at what time.

In previous waves of CDS, all of these tests were programmed and worked properly. For the recent wave, we decide to modify the programming so that it is modularized and can be easily understood by another programmer if a problem occurs and needs to be fixed. Time was spent to ensure that one test wass implemented correctly, and we then applied the method to the other two tests. The core technique is to eliminate the hard code and use one procedure for basal and one procedure for ceiling calculations. To be able to call the procedures in different tests and different items, parameters are passed into the procedure. In the end, the basal and ceiling procedures were called as many as eighty times. Additionally, both the PCG and Child instruments were able to share the same code with very little modification. This method was much appreciated in terms of maintenance and troubleshooting by the project team

6. Conclusion

CDS 2014 is a complex study. Several challenging features were specified by the research staff in order to assist interviewers in easily navigating the various data collection instruments while collecting quality data. With careful design, testing and implementation, new features were developed to program three high quality instruments, each of which is well accepted by the research staff, project team and interviewers.

From our experience, Blaise proves to be a powerful and flexible survey program system and helped us to achieve almost all complex requirements for this study.

7. References

Blaise 4.8 Online Assistant – Statistics Netherlands

CDS 2014 Study Guide – Survey Research Center, Institute for Social Research, the University of Michigan