Longitudinal Survey Data – "Move It Forward"

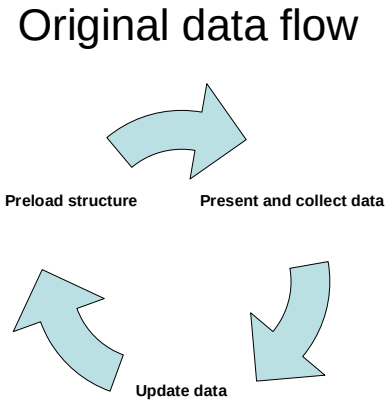
Rhonda Ash and Danilo Gutierrez, The University of Michigan

HRS Overview

The Health and Retirement Study (HRS) is a complex longitudinal survey covering a wide range of health issues and economic concerns related to aging and retirement. The average interview lasts more than an hour and is administered to over 22,000 respondents. The HRS began in 1992 and had been carried out for three waves at two-year intervals before it was merged in 1998 with its sister study AHEAD, which covered an older birth cohort and somewhat different content area. At that time two new age cohorts were added to fill in the age range above age 50, and a "steady-state" sample design was born which would maintain representation of the over 50 population by bringing in a new (younger) age cohort every six years. That change dramatically increased the ability of HRS to address the effects of events and policy changes within its scope and allows HRS to aspire to a very long lifespan with continual expansion and adaptation of its content as these changes occur.

HRS Preload Process - History

With a longitudinal survey about finance, family, and health, data collected from a prior wave could simply be presented and confirmed or updated, vastly reducing the time needed for the next interview. HRS developed a structure to house thousands of variables that were pulled forward from prior waves. As we were learning Blaise and the program's early version constraints in 2002, we made decisions about the preloaded data structure to attempt to ensure the program's performance would not be degraded. We developed a schema whereby the data was loaded into one location (structure) and referenced or updated in that location. As data was presented and changed the original location data would be changed. Hence, the term "preload" quickly became a misnomer.

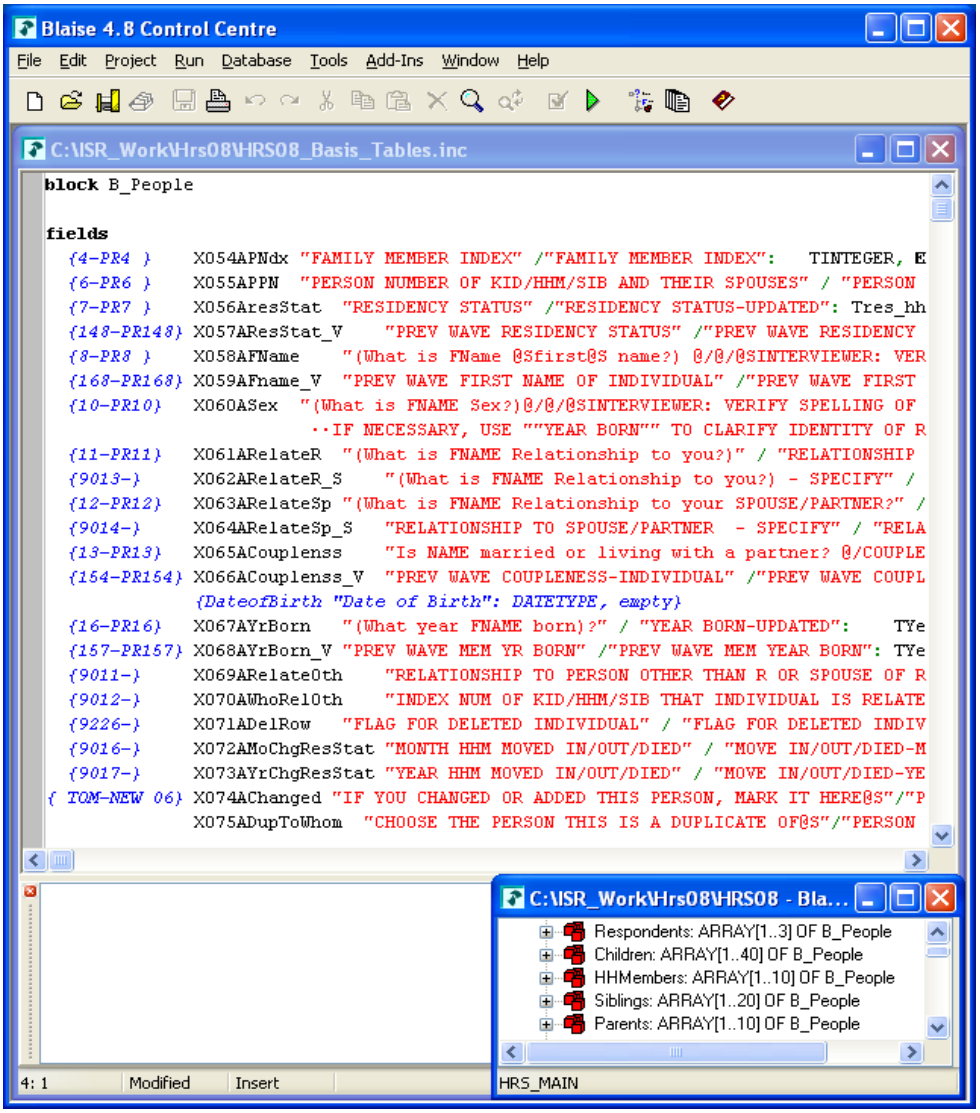


Early Blaise Restrictions

With an application the size and complexity of HRS we had to be very concerned with the amount of stored data that we carried in the instrument. Early in the process of working with Blaise, HRS ran into problems which caused us to be cautious about the number of fields in our very large instrument. The more fields we added to the route, the slower the application became until it failed -- or the RULES would not compile at all. Since the preload structure contained a lot of fields, a scheme with a single copy of preload which would be updated as the interview progressed was baked into the design early. While learning Blaise, we quickly designed the original "write it back" process. A block of 25 fields that

contained only people data was created in the coverscreen section of the instrument that collected such key information as respondent's name, gender, coupleness, and age. This block was replicated 83 times for the different types of people:

3 Respondents (current Respondent, current Spouse or Partner, any new Spouse or Partner)
40 Children and spouses with room for any new children if the Respondent recoupled
10 Household members who are not offspring
20 Siblings and
10 Parents



For certain variables we needed to compare the previous wave's statuses to the current wave -- for instance, married before (*X066ACouplenss_V)* versus married now (*X065ACouplenss)* -- in order to determine which questions or what version of a question to ask. As a result, we had to duplicate several fields in this preload structure and ensure that we never updated them. For example *X065ACouplenss* was updated, to reflect current marital status, whereas *X066ACouplenss_V* contained the “virgin” preload value and *X066ACouplenss_V* was never updated.

Blaise Selective Checking and Gates

Certain aspects of Blaise, while helpful for many situations, presented certain challenges when dealing with the HRS coversheet's complexity. Some of these aspects are described here.

Selective checking: Blaise documentation stresses that

> **"**In a large instrument, there may be thousands of questions and thousands more rules that govern their implementation. In such a data model, the constant checking of all rules could overwhelm the computer. In order to know which rules to enforce, Blaise employs a selective checking mechanism based on parameters and blocks. Blaise uses parameters to optimize the performance of the checking mechanism during instrument use. By keeping track of parameters, both explicit and internal, Blaise knows which blocks to check."[1]

As the interview progressed, Blaise's selective checking mechanism would rerun the rules that were active "on the route" from the beginning of the application to the current field being asked. If we were updating fields that had already been on the route, that set of rules was rechecked.

Gates: Code that would stop the selective checking was added to the application to stop the process in certain conditions in order to take a field or block off the route.

KEEP statements: Used to maintain the data when a field is no longer on the route. When a field's value changed, other fields may be taken off the route; without KEEP statements the data in those fields are lost. With a different way of programming, we could dramatically reduce the number of KEEP statements needed.

Quite often we would add a field whose only purpose was to lock off other fields from the selective checking. Once we reached a point in the application, a field was presented and once answered, the logic of the RULES would remove fields or blocks from the route.

```
If A047_ = empty then
    Call Thatblock of fields
Else
   ThatBlock.KEEP
Endif
```

A question as simple as "Are you married?" became a programmer's challenge as the marital status changed and was written back to the original "Preload" field.

In this example, *Respondents[1].X065ACouplenss* was preloaded with a value of MARRIED. As the interview's coversheet progresses, we ask if they are still married to the same person at *A020TSameSpP*. If the answer to *A020TSameSpP* is NO, the value of *Respondents[1].X065ACouplenss* is changed to OTHER and the selective checking mechanism alters the route. Now *A026_Rmarried* shows up on the route. (*A026_Rmarried* asks a single respondent or a previously married respondent who is no longer married to the same person if they are now married to someone else.) If *A026_Rmarried* is answered YES the status of *Respondents[1].X065ACouplenss* becomes MARRIED once again. (Still with me?)

Part 1:

[1] Excerpts from Blaise 4.8 Online Assistant

```
IF (Respondents[1].X065ACouplenss= MARRIED OR
        Respondents[1].X065ACouplenss= PARTNERED_VOL))   THEN
        A020TSameSpP_A.Keep   {Keeps the data values when the field goes off the route}
        A022_.keep
        IF A020TSameSpP_A = empty then  {only ask if empty}
            A020TSameSpP_A {Is Clara still your wife?}
           If A020TSameSpP = NO then
                Respondents[1].X065ACouplenss := OTHER
           Endif
        Endif
Endif
```

Part2:

```
    IF A026_Rmarried <> empty {stop lock out when X065 gets assigned below}
    or
      ( piBA_HHX024_ReIwHH_V = yes and
        (Respondents[1].X065ACouplenss= OTHER OR
        A020TSameSpP_A = NO )
      )  THEN
         A026_Rmarried
        If A026_Rmarried = Yes then
           Respondents[1].X065ACouplenss := MARRIED
        Endif
```

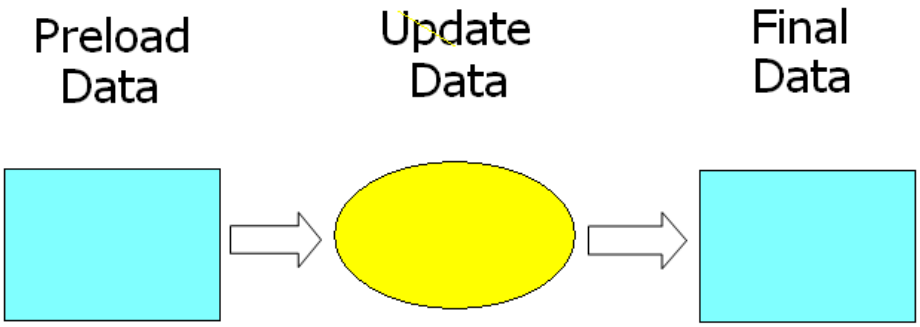
How To Fix The Problem

One of the more difficult issues with the "old" structure is that the selective checking mechanism unexpectedly changes the data values of fields on and off the route by re-evaluating the rules. After several waves of adding gates and reorganizing the code, the application became error-prone when new content was added or old content changed. In the example above, we referenced the *Respondents* block which is an array of three instances (current Respondent, the Spouse or Partner, new Spouse or Partner). We would load data from the prior wave into that arrayed block, and then alter the values as we progressed through the application.

To change that process we adapted a "Move it forward" philosophy. That is, once the data had been set into any field we would not re-assign it. We would move the data forward into a block or field that had not been altered.

We now have the same arrayed structure, but added a new location where the data values will not be changed. The prior wave's data is currently loaded into the *PRELOAD_Respondents* block. Any updates to the preload are recorded in the *SecA.Respondents,* , and the updated data are copied to the *Respondents* block. The *PRELOAD_Respondents* data are never altered after the data loading process, thus the Blaise selective checking mechanism is not triggered. The same philosophy is now followed for the *Children* data: we use three arrayed blocks for the preloaded, updated and final data. All updating is done in the coversheet section (*SecA*). Any reference past this coversheet section would look at the data in the final copies of each structure (*Respondents or Children).*

Move it forward



Since it is important to maintain comparability in the instrument across waves with a longitudinal study, the thought of restructuring is nerve-racking. The data being collected must be able to be connected to the prior wave data, and be assured that the collection process does not change the meaning of any question's intent. The changes made were in how the data are stored, not in how the questions are asked. We are now able to store additional data structures for the *Respondent* and *Children* blocks. Restructuring allowed us to only alter the data in the update data locations and reduce or eliminate the back-writing. The code became easier to manage and understand. What took many lines of code before now takes only a few lines. We now need less code to handle the selective checking mechanism.

```
IF Preload_Respondents[1].X065ACouplenss = MARRIED OR
    Preload_Respondents[1].X065ACouplenss = PARTNERED_VOL   THEN
          A020TSameSpP_A

IF Preload_Respondents[1].X065ACouplenss = OTHER) OR
    A020TSameSpP_A = NO       THEN
          A026_Rmarried
```

Now, since *Preload_Respondents[1].X065ACouplenss* is never altered, we no longer need to protect against the selective checking mechanism. The rewrite gives us only one location of data to reference in future sections. The final data location helps to simplify and avoid interrelated problems between sections.

```
Respondents[1].X065ACouplenss :=  A038TCouplenss_A
```

Conclusion

Making any changes to our instrument, even small ones, was a matter for grave consideration because we never knew what unintended consequences might emerge. As a result, often times sometimes desired changes to improve question objectives were not implemented, especially if they were discovered late in the program development period.

Why was the coversheet rewritten?

- It was very difficult to manage code.
- Anticipating how the selective checking mechanism would behave became highly problematic in our very large instrument.
- The code became hard to understand especially for non-programmers.
- With any change, the whole instrument needed to be fully re-tested.
- We would have to check to make sure that data was not dropped "off the route" and lost.
- We had to make sure we did not unintentionally overwrite data.
- There were many interrelated problems between sections with the way the code was written.
- Changing the route in the coversheet adversely affected the logic in later sections. These effects were observed in other family data sections.

During the design of the re-write we considered these major issues. Each of these issues was examined and addressed. With the investment of time and planning in this re-write, the efforts will pay off into the future. The benefits have already been evident beyond our original hope. Now, with the easier code readability, non-programmers are able to look through the code and better understand the flow of the data collection. We are also much more confident that we can modify those areas of code without introducing unintended effects and endangering the whole instrument's integrity.

With the benefits that we have received from this investment, we plan to re-write other sections of the application. We have already applied this schema to the section that collects sibling data. Next, we plan on re-writing the section that collects children transfer data.

References

Statistics Netherlands, Blaise Department, Blaise 4.8 Online Assistant.