Part 5 - Converting HRS from Blaise 4 to Blaise 5 – the Example of Rosters

Jason Ostergren, Survey Research Center, University of Michigan

1. Background

Historically, The Health and Retirement Study (HRS) has had a handful of cases where it presents arrays of people or pensions in table form in Blaise 4 versions of the instrument. These tables can be filled with preloaded answers from prior waves for confirmation, and also must allow for the addition of new rows to accommodate people or pensions that are new or that previously were missed. These include children and household members of the respondent, which are asked about in separate tables near the beginning of the survey. The information gathered there is used in many subsequent sections - for example, to drive loops and fills about money transfers or as options in enumerated type questions about help provided. HRS later presents a similar table about siblings of the respondent, used in similar ways. Finally, HRS has a table for pensions and associated employer information. That is crucial for setting up the lengthy sequences devoted to understanding respondents' pensions and gathering related employer info to verify information in some cases. Pension questions often seem very difficult for respondents because the subject matter is often not well understood. The survey design must do as much as possible to avoid adding additional complications due to user-friendliness issues. People rosters seem comparatively easier because respondents face fewer conceptual problems in answering these questions, but we have found that it's not uncommon for things to go off track even there, especially when the survey is not easy to use. The main HRS survey has always been interviewer administered, so while user friendliness issues may cause confusion and delays, the interviewer has always existed as a backstop who can help get things back on track most of the time. HRS 2018 includes the first self-administered interviews in our main survey, however, and that has required a stepped up effort to make sure that these questions are presented effectively.

2. Rosters – more detail

In each case, these sequences are built around underlying arrays of blocks, which may be preloaded before the survey starts by external tools, and which are passed on to later sections of the survey for gathering additional data in follow-ups or for organizing later sequences. In general, the array works as an input to these sequences, and there is usually a matching array that constitutes the updated output for later use (we don't simply write back to the original, both because there is sometimes need to compare and also because we have found that writing back in Blaise usually produces complications). This array data is meant to be concise and to transmit only the required information between waves and between parts of the survey. It is not organized in a way that is optimal for presentation as questions. For that, we populate an array of blocks that contains this basic data as well as fields which allow it to be confirmed or collected in more natural questions. For example, we may have a single variable containing compound coupled status in the preload array, but we add additional variables in the asked array to allow for a sequence of questions asking first about continuation of any known prior relationship, then marital status, then non-marital partners. Data is initially unpacked from the compound variable in order to prefill answers in the split ones and then it is re-packed into a corresponding compound variable in a new copy of the preload array for use in later sections. The preload array also contains variables which are essentially administrative (such as tracking numbers) that do not have corresponding asked questions, but must be passed along to the output data.

3. Rosters – the solution

We have chosen to present these arrayed questions for asking grouped on screen as a table or grid rather than simply a looped series of questions. The reasoning behind this is twofold. First, we think that this presentation of the looped data allows for better situational awareness about the whole set being asked

about. Though small in number, we have historically had a great deal of trouble reconciling duplicates and other anomalies in these arrays. We believe that the likelihood of problems would turn out to be much greater if the screen didn't show, at a glance, which people or pensions were already accounted for. Secondly, we want to make it easy to return to an earlier item in the roster to correct mistakes. An alternative which required walking back through every question across multiple items has always seemed like an invitation to mistakes. In Blaise 4, the Table construct made it fairly easy to use arrow keys to move to previous items without touching each intervening question and, crucially, to be able to see how to get there. Due to the interviewer-administered nature of our Blaise 4 instruments, it was also possible to cram all of the asked fields into the table on one screen by careful management of column width. In Blaise 5, we were looking to retain an overview and navigability, while actually only displaying one question (as is our standard practice in Blaise 5) or one rows' worth of questions per screen.

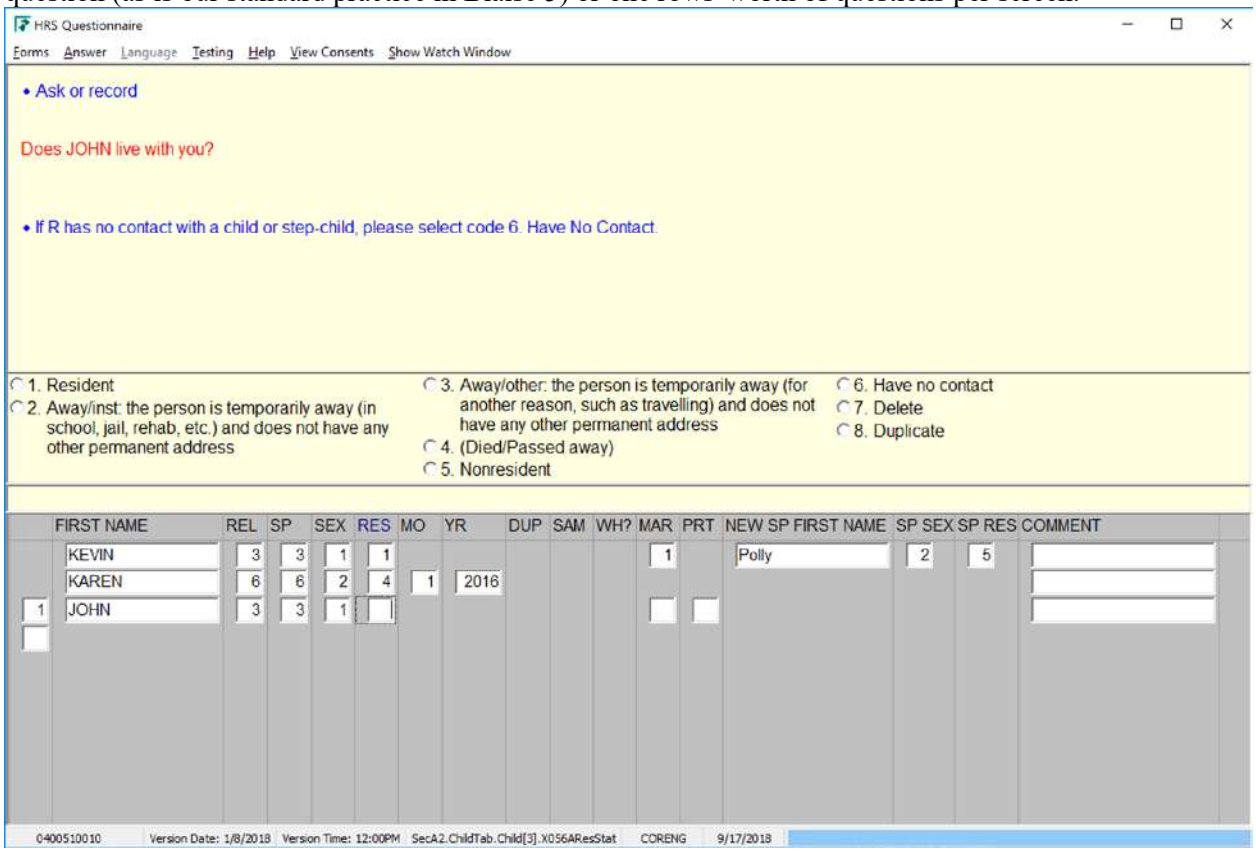

Figure 1. A child roster in the Blaise 4 HRS instrument.

Our new plan was strongly influenced by one of the methodologists here who leaned in the direction of using dashboard-type screens for the new design in Blaise 5. Roughly, the notion is that you start with an overview screen on which each item (person or pension) is represented by a clickable button or link which would bring up another screen where editing could take place or new questions could be administered. Completing the detail screen would then automatically return you to the dashboard, with some indication of which item had just been completed and of which ones still needed to be examined. Once all items were marked as examined, the button to allow movement past the dashboard would finally appear. We made some efforts to adopt this idea directly, which will be mentioned briefly below. However, in the end, we effectively reversed it, essentially going through the detail screens one by one and placing a stand-in for the dashboard at the end of the sequence in the form of a summary screen, which will be discussed in detail below.

Because we had some experience with customizing the page handler in Blaise 4 IS, and had also made early efforts to get into customizing the Blaise 5 browser data entry client (while it was in the older Asp.Net architecture), our first attempt to implement new rosters involved customizing the data entry client to add a roster page built entirely from our own javascript code. (HRS has adopted the practice of referring to all data entry customizations as "routers" which will continue below for brevity's sake.) This turned into a working prototype of the pension roster which showed a partial grid vaguely like our Blaise 4 design, but emphasized manipulation of the line items in the grid with obvious visual cues for adding, deleting and editing. In addition, detail screens were in-page modal boxes with a set of related input boxes (in this case employer details in a form layout). The general notion was that data would be loaded into the javascript from the preload array, would be manipulated entirely within the javascript, and then saved to the post-roster copy of the preload array similar to the way it was handled in the past. While on the screen, user interaction would not need to be constrained by the Blaise rules engine or layout schemes, making it easier to potentially add manipulations like drag-and-drop functionality, for example. In the end, concerns about whether the plan would be very future-proof caused us to shelve it, and that turned out to be prescient. Originally, we held out hope that we would be able to do both our interviewer-administered and self-administered surveys in the browser client and we thought that the early work we did would be usable in some form in the version we fielded. However, we ultimately were asked to split our work between Windows DEP and browser DEP, which would have required two separate routers, doubling the workload. In addition, the performance advantages of the MVC architecture change which arrived late in the game (or rather performance downsides of the Asp version) would have forced us to abandon any code based on that original design anyway. In the end, we decided to rely much more heavily on out-of-the-box capabilities of Blaise 5 to build the rosters, but the lure of the design freedom offered by the router approach will likely cause us to revisit the idea when we start looking at the next wave of HRS.

HRS ultimately found a compromise solution, which adopted a detail-page summary question strategy that had previously been used only for (the more conceptually complicated) pensions, but now would be used for all rosters in the new survey (our terminology can get confusing here: the "summary" question is a sort of shortcut on a detail page, but the "summary" screen which follows all the detail pages aggregates all the roster information on one table for group review). The original idea from our Blaise 4 instruments was was that the respondent would be read a description of each item preloaded from a prior wave (a 'summary' of each item, which might sound something like: "you had a 401K plan from Ford, where you worked from 1980 until 2002, that you called your 'Ford plan'") and then allowed to confirm that nothing had changed with this single answer (thus skipping the remainder of the detail questions), or to deny that the information was correct and edit the item in detail. Following the summary questions for preloaded items, the respondent would get asked if they wished to add a new item as many times as needed until they said 'No' at which point the loop and thus the questions would end (there is a maximum number of items specified, but it is set considerably higher than the largest amounts we have gathered across many waves). Once all the identifying information had been collected or confirmed, the respondent would immediately enter a new loop through the same items for a series of follow-up questions (a short series in the case of people and a very long series in the case of pensions). Other than layout, the major new component in the Blaise 5 design was a summary screen between these two parts, allowing final validation and navigation for corrections and acting as a gate, closing of the task of compiling what was essentially a full list of identifiers.

This came about in part because HRS put a lot of effort into avoiding the need for significant vertical scrolling and eliminating any possibility of horizontal scrolling. In most of the instrument, we present only one question per page. We had to do a fair amount of experimentation to see what questions could practically fit in a detail screen, and that naturally pushed us towards splitting up (in the manner previously described) questions that in prior waves had fit in one table. We also were pushed in this direction (splitting the old table) by the need to improve the clarity of questions and answers for self-

administered respondents, since prior designs had sacrificed ease of understanding in order to cram more question-columns into the Blaise 4 roster.

We finally landed on a conceptual basis for this split as follows. One of our co-PIs did an analysis of the frequency of changes in status between waves among the various roster questions. She proposed that we design based on a distinction between variables that change more rarely or due to mistakes (such as first name), and those that might be expected to change, perhaps more than once, over multiple waves (such as marital status). This happens to map on to the fact that the former may be of lesser (or sometimes, no) use to users of HRS data (they exist more to allow tracking and identification internally). Additionally, we not only have immediate follow-ups where keeping track of which person or pension one is referring to may be tricky; we often return to the same set of people later for follow-ups in other sections covering different content areas. So the first set of variables constitute a sort of contract between the respondent and interviewer and/or survey designer on a common reference set for immediate and much later follow-ups.

In Blaise 5 what this looked like was a separate single question screen for each person or pension, making it easy and clear for a typical respondent who simply needs to verify that we have the correct set of children, pensions, etc. The implementation was tricky, however, as we wanted the detail questions to appear on the same screen if the respondent answered 'No' to the summary/verification question. In other words, the respondent should initially see a question asking if a set of information is correct; upon answering "No, it needs editing" and clicking the "Next" button, we want the respondent to see the editable questions appear below on the same page. This was a difficult proposition since HRS had set a requirement that nearly all fields would allow "Empty" in our self-administered version, meaning that the engine is prone to skip right past a bunch of empty questions; not to mention that frequently these questions will be pre-filled by preloaded values. Blaise 5 offers ways to force routing checks within a browser client page, but when we looked at them, we decided they wouldn't be easy to maintain in our complicated loops due to overhead in layout. That is, HRS is too big to practically use the Blaise 5 layout editor (the "Layout" tab while the .blax file is open), we have multiple programmers working on the same survey, and we use define statement to remove parts of the instrument from testing as needed which has the side effect of ruining any work spent on the layout editor by permanently eliminating the temporarily removed bits. We were able to achieve this desired result unexpectedly with simple routing through code alone, with the quirk that the final question on the page had to be an enumerated type. We assume that underneath, Blaise 5 is failing to leave the page due to the appearance of more on-route questions and this works fine for us, but this may prove to be a brittle solution in the long run if it turns out that we are exploiting a mistake in Blaise.

```
3359        IF X055APPN = EMPTY THEN
3360           A208ANewPerson
3361        ELSEIF A221AConfirm <> NO THEN
3362           IF SELFADMIN AND X058AFName <> MISSINGNAME AND X058AFName <> MISSINGNAME_SPN THEN
3363              A221AConfirm
3364           ELSE
3365              A221AConfirm := NO
3366           ENDIF
3367        ENDIF
3368        IF A221AConfirm = NO OR A208ANewPerson = yes THEN
3369           GROUP_EditChild_GROUPTEXT
3370        ENDIF
```

Figure 2. Simplified code snippet for child detail summary page.

```
<RouteItemLayoutInstructions RouteItemName="SecA2.ChildTab">
  <Instructions>
    <GridInstruction Locator="Before" RouteItemsPerPage="3" />
    <TemplateInstruction Locator="Before" TemplateName="HRS_SectionIndex" TemplateTarget="MasterPage" />
  </Instructions>
</RouteItemLayoutInstructions>
<RouteItemLayoutInstructions RouteItemName="SecA2.ChildTab.GROUP_CHILDGATE">
  <Instructions>
    <GridInstruction Locator="Before" RouteItemsPerPage="1" />
    <TemplateInstruction Locator="After" TemplateName="HRS_MasterPage" TemplateTarget="MasterPage" />
  </Instructions>
</RouteItemLayoutInstructions>
```

Figure 3. Entries for switching master page TemplateTarget and RouteItemsPerPage in the .layout file (manually added).

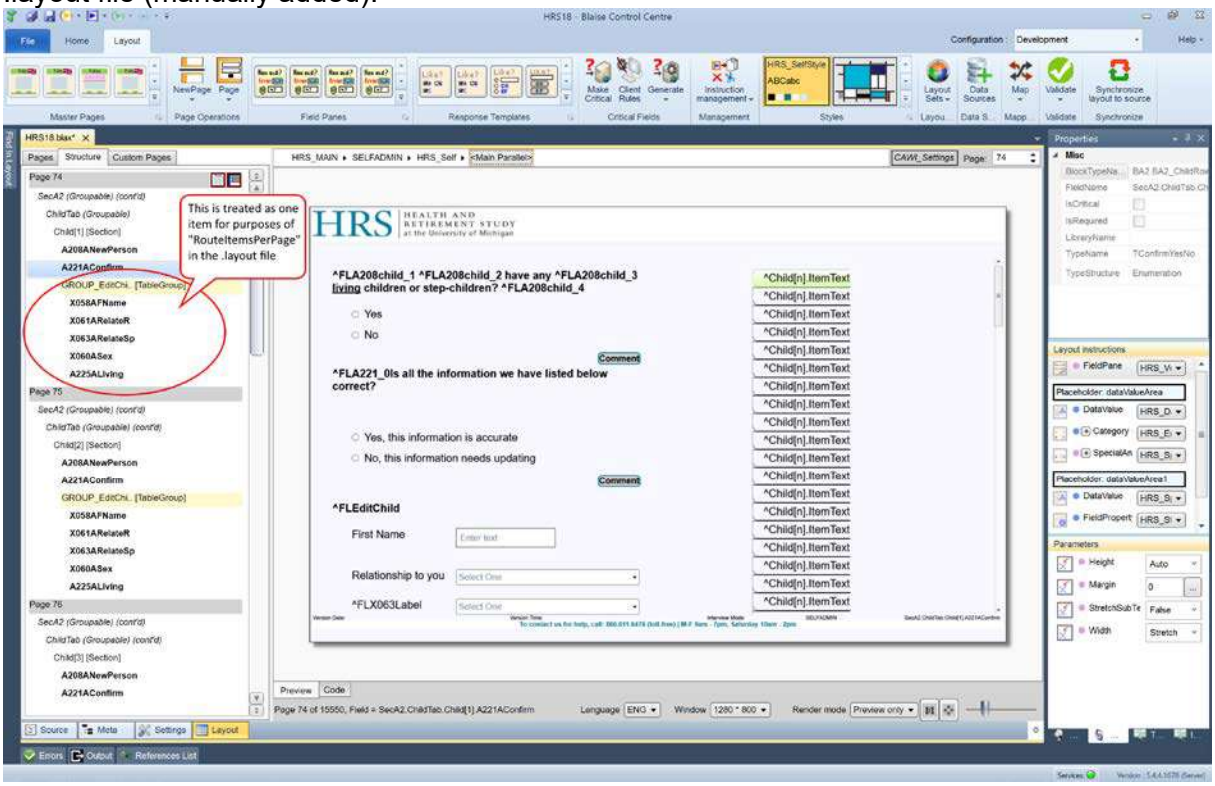

Figure 4. Even though we do not use it for editing, we can however use the layout view to see edits we make manually to the .layout file (namely switching to 3 questions per page in the detail sections) – the above screenshot shows the detail summary and detail question set page for one child.

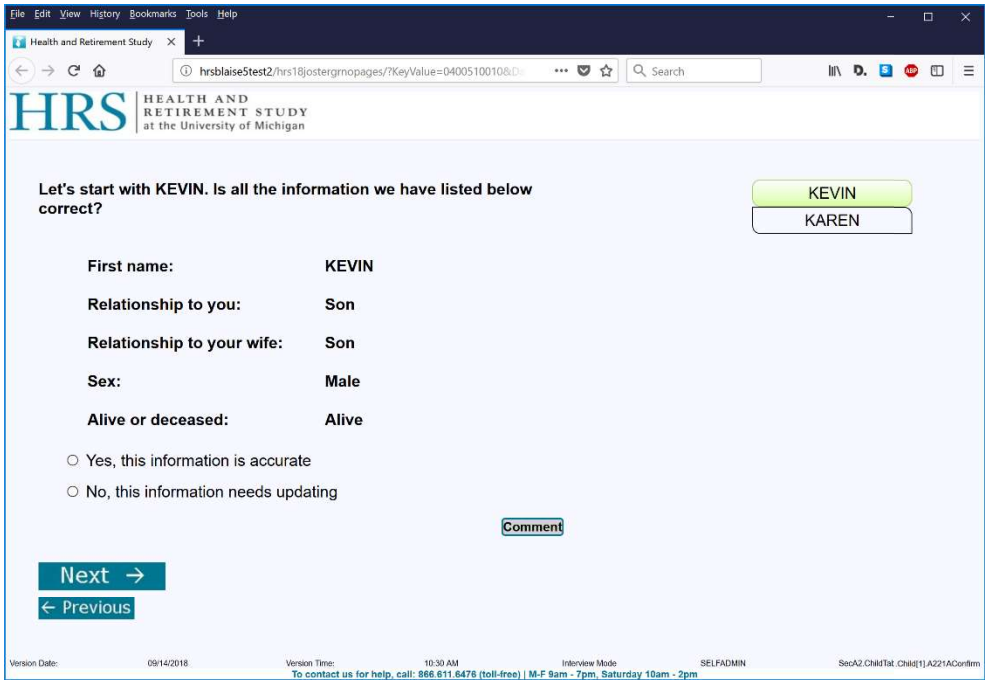

Figure 5. This is what our child detail summary screen looks like for a preloaded child reported in a previous wave. Note the names on the right are clickable navigation buttons and the current child is highlighted.

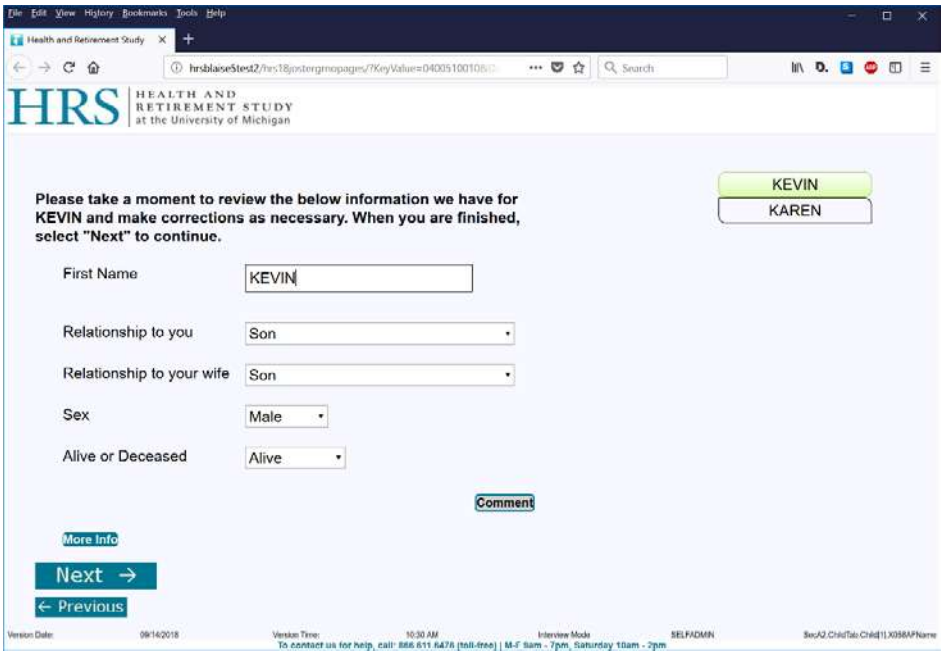

Figure 6. If the detail summary question is answered "No", after clicking "Next," it is removed from the route and this group of detail questions appears in its place. If it had been answered "Yes" instead, the survey would have proceeded to the next child detail summary page.

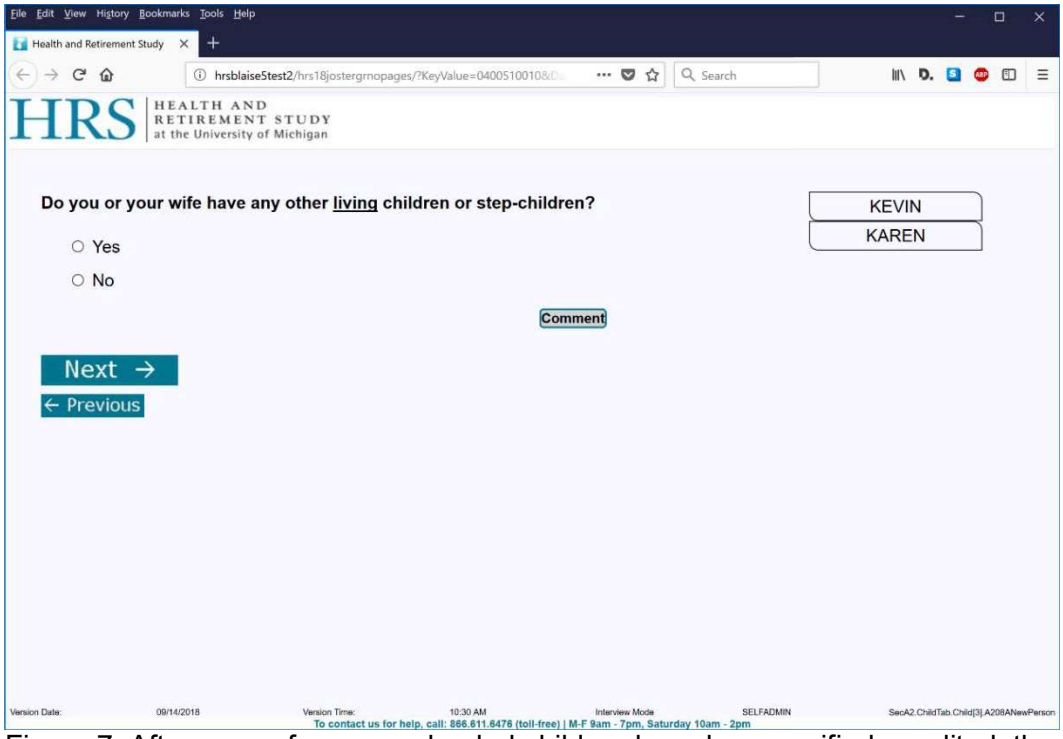

Figure 7. After pages for any preloaded children have been verified or edited, the survey displays pages allowing new children to be added. Similar to the detail summary question, one answer ("Yes," in this case) opens up a group of questions (see Figure 5); the other answer proceeds to the roster summary screen. Note that no button on the right indicates the current page at this stage.

Because HRS had been unable to make a satisfactory dashboard design work, we opted instead for a roster summary screen following all of the verification, new item and detail questions. The summary screen needed to be able to return the respondent to any individual item if they discovered a corresponding mistake in the summary. In addition, when on the individual item screen, it needed to be possible to return to the summary screen with one click (i.e. not by advancing page by page back through multiple potential items). We also already wanted to have the list of identifiers on the screen so the respondent was able to see who they had added and navigate backwards even before they reached the roster summary. Finally, the one click roster summary button needed to be hidden until they actually arrived at the roster summary (which was a problem to implement for a long time, but finally solved using the 'IsVisited' property). These navigation buttons were originally placed on the left of the screen so that they would catch the respondent's attention. Later we moved them to the right side in order to privilege the question itself. In the end, we came to see the buttons less as navigation options for the first pass through and more as a list to help with workflow (and one that updates if the names are edited or new rows are added). Once the respondent reached the summary screen, instructions would then point out the navigation buttons because they would be necessary to reconcile problems found at that stage in an efficient manner.

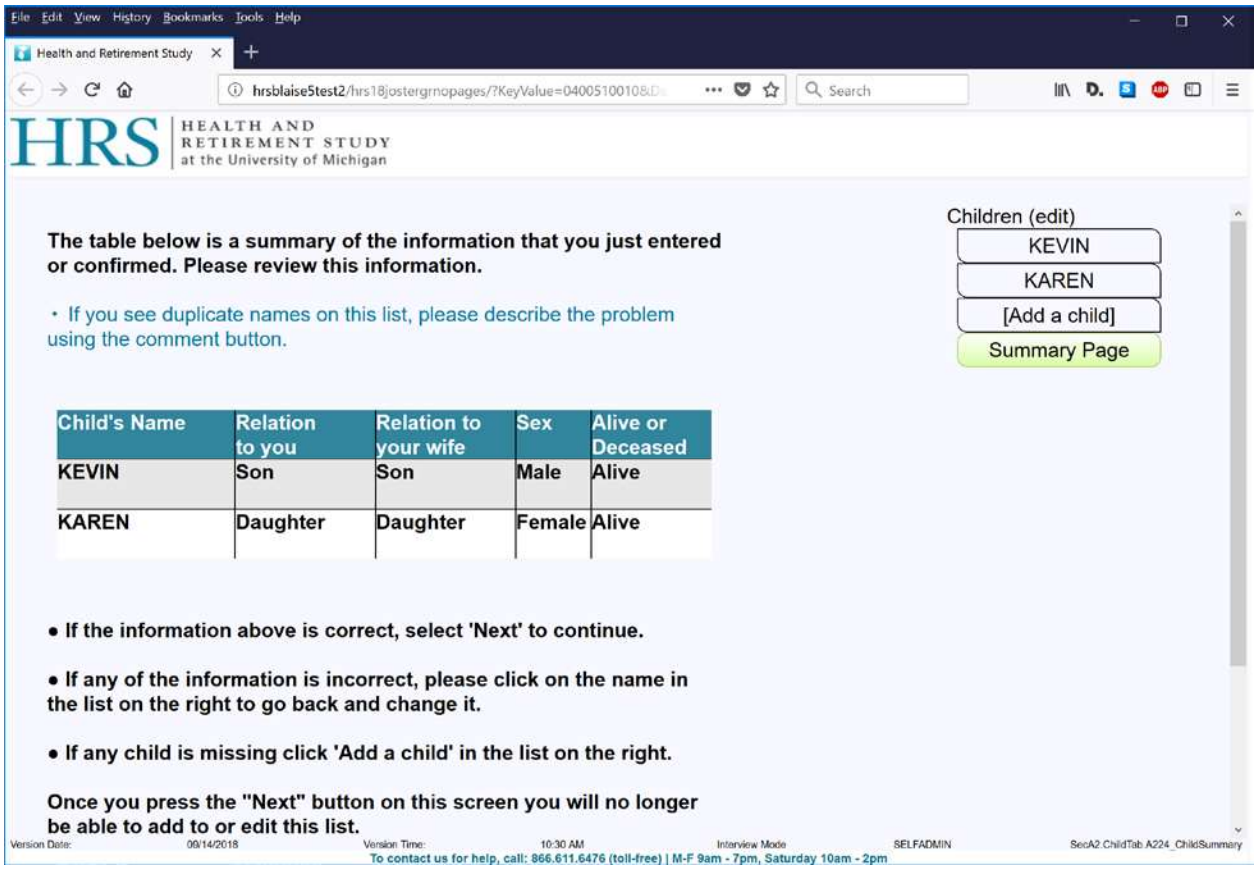

Figure 8. The roster summary screen. Despite our efforts, the "Next button is just off the screen to the bottom, necessitating some vertical scrolling. Note that, having arrived at the summary page, the buttons for "Add a child" and "Summary Page" have appeared in the list on the right (the buttons were always there, but they were hidden until the "IsVisited" property of the summary page was true). After this point, if the respondent goes back to earlier pages, these extra buttons remain visible to facilitate easy navigation.

HRS chose the Section Index Page template pattern in the Resource Database for its roster design as it allowed for normal routing through the questions, while at the same time providing an alternate means of quick navigation. Parallel blocks would have been an alternate possibility, but the Section Index templates ultimately worked out a bit better. As is often the case with our more complicated sequences, we had to go against the grain of what seemed to be intended. The section index templates provide the quick button navigation we need, but appear to be intended for large "sections" of a whole survey, not single pages. In other words, under the hood there appears to be one big section index list for the entire survey – you can't define multiple section index lists. As a result, we have to gate off each roster from the others or else buttons for previous rosters appear in later ones. Gating off rosters and other sequences is not new to HRS, but we had been trying to move away from doing that for the sake of self-administered surveys, since gates might prove confusing to respondents trying to navigate the interview by themselves. Gating off the rosters caused some new additional problems as well, which will be described below. However, since most of the functionality for the navigation buttons is built in, we had relatively little trouble adding this functionality on top of our existing template designs in the resource editor.

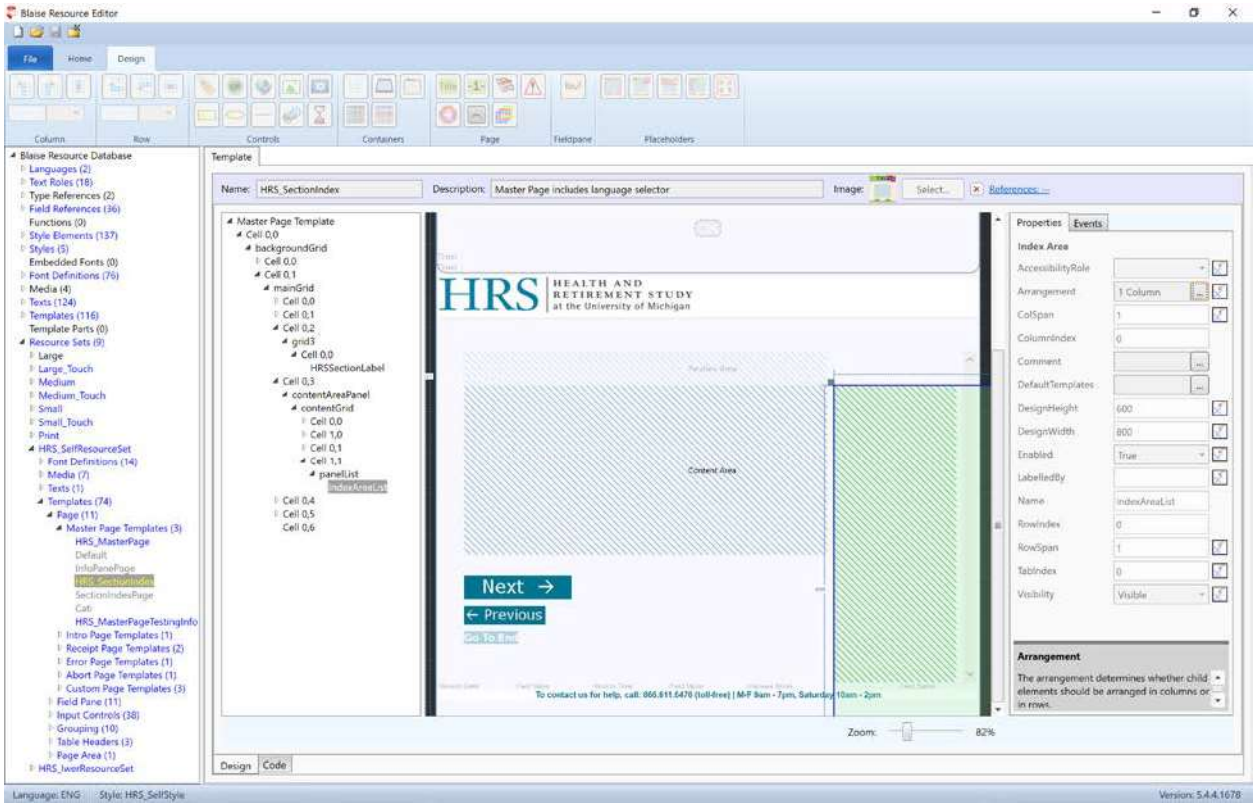

Figure 9. The IndexAreaList master page template.

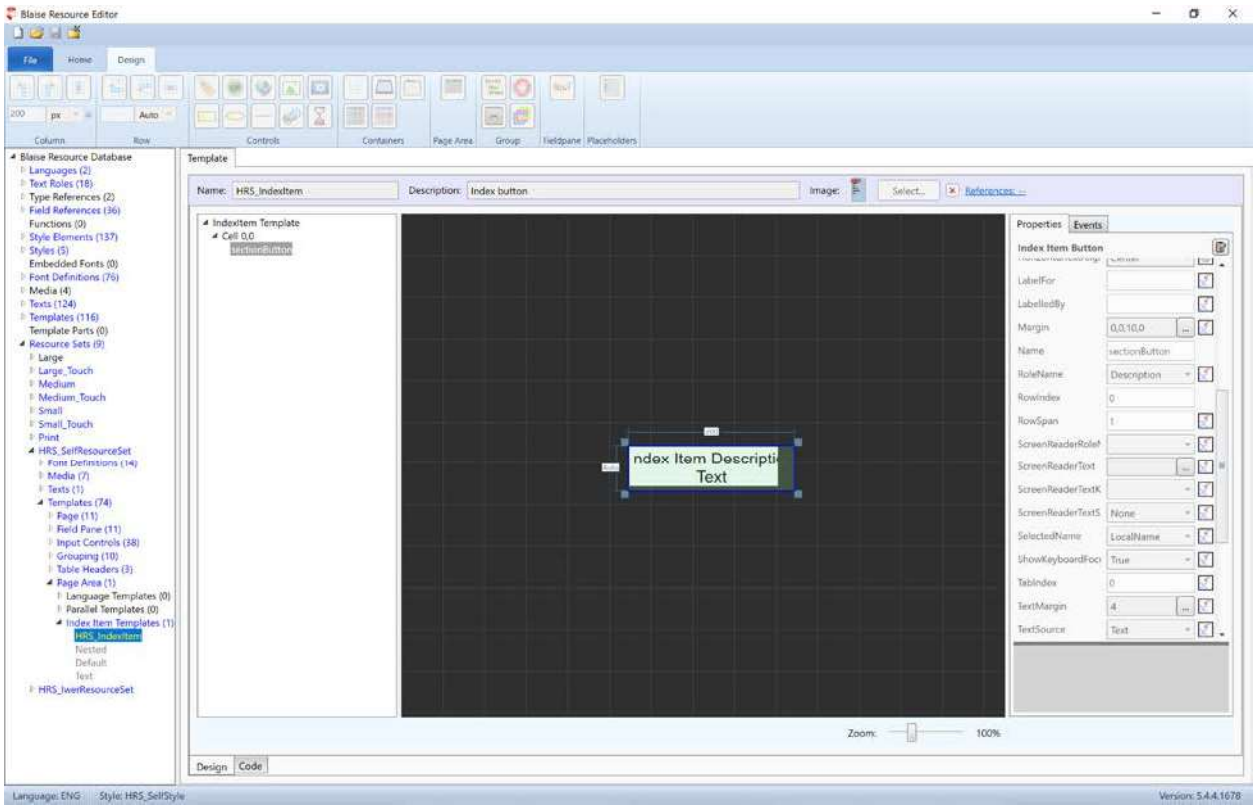

Figure 10. The Index Item button template.

Gates posed one particularly big problem that resulted in a lot of spinning of wheels on solutions that never worked out. We want a gate to trigger based on a screen outside of a loop (basing it on a loop screen poses its own special set of problems because you can't predict at design time exactly where the loop will end in a session). Typically such a screen is a summary or an informational screen with no answer items, only a "Next" button (and "Previous" button). In Blaise 4 and in the interviewer-assisted mode of HRS in Blaise 5, these screens would have a simple codeframe with only one option ("1. Continue") and the "noempty" attribute ensured that a value of "1" would always be entered. This was important for later routing that depended on this gate being closed and it was important for external tools which HRS uses that depend on the value being recorded in the audit trail as well. However, self-administered interviews in Blaise 5 were beset with a number of problems in this case.

First, no answer item would be presented to the self respondent both because that represented an extra burden and because the requirement that every screen allow empty meant that it was not going to be possible to police whether the answer was actually given. Inconsistent answers on this screen would make the gate useless. What we wanted was to assign "1" to the gate question when the "Next" button was pressed, without even showing the respondent the "1. Continue" code in the first place. This turned out to be a fairly easy thing to do when it came to the "Next" button itself. However, the respondent can also advance past the screen in other ways, such as swiping, or pressing enter (we do want to support keyboard use for respondents who may have trouble with a mouse). There is no provision in the resource database for attaching actions to pressing enter to leave the page. What this meant is that a self respondent pressing enter on that screen would fail to close the gate, which would result in many incorrect skips. In the end, with direct help from CBS, we found a fairly convoluted workaround to this problem which would trick Blaise into making the assignment. This is where we discovered the "IsVisited" property, which was part of the solution.

```
4550      A224_ChildSummary
4551      A224_ChildSummary.AlienActionEvent := '1'
4552      IF A224_ChildSummary <> 1 THEN
4553         A224_ChildSummary := FUNC_SetGateFieldsOnPreviousFieldIsVisited('A222_ChildGate', A224_ChildSummary.IsVisited)
4554      ENDIF
4555      A222_ChildGate
4556   ENDGROUP
4557
4558   FUNCTION FUNC_SetGateFieldsOnPreviousFieldIsVisited : INTEGER
4559   PARAMETERS
4560      IMPORT
4561         piCheckCurrentField: STRING
4562         piTriggerFieldIsVisited : TIsVisitedFieldProperty
4563   RULES
4564      IF POSITION(piCheckCurrentField, CurrentFieldName) > 0 AND piTriggerFieldIsVisited = Yes THEN
4565         RESULT := 1
4566      ENDIF
4567   ENDFUNCTION
```

Figure 11. Simplified code snippet for gate.

This solution correctly made the assignment whether the “Next” button or the “Enter” key was used, but our troubles were not over yet because it failed to record the assignment in the audit trail. In the end, we solved this problem with a router in the new MVC architecture which we had adopted by that time. In short, we overrode the nextPage and nextField events (which covers both leaving the page via button and enter key) to check whether a specific property was present. If so, it would add an event to the audit trail that would account for the assignment, which we are able to read later from our tools.

```
protected nextPage(): IActionKeyValuePair {
    this.signalSupressService.suppressSignals();
    let nextPageResult = super.nextPage();
    if (nextPageResult) {
        this.alienActionEventService.checkForAlienActionEvent(0);
    }
    return nextPageResult;
}

protected nextField(actionObj: IAction, controlId: string): IActionKeyValuePair{
    let nextFieldResult = super.nextField(actionObj, controlId);
    if (nextFieldResult) {
        this.alienActionEventService.checkForAlienActionEvent(5);
    }
    return nextFieldResult;
}
```

Figure 12. Code snippet triggered by both events which can cause a forward page change from custom-action.service.ts, a new ActionService.

```
public checkForAlienActionEvent(kind: number) {
    if (kind === 0 || kind === 5) {
        const actionNameAssignField: string = 'AssignField';
        const fieldPropertyName: string = 'AlienActionEvent';
        const actionKind: string = (kind === 0) ? 'NextPage' : 'NextField';
        const detected: string[] = this.detectFieldProperty(fieldPropertyName);
        if (detected.length > 0) {
            this.auditTrailService.logActionEvent(actionNameAssignField, detected.join(','), AuditTrailLevel.None);
        }
    }
}
```

Figure 13. Code snippet writing an extra record to the audit trail service when triggered on a field with the correct property from alien-action-event.service.ts, another new class.

4. Conclusion

The development of these rosters for HRS in Blaise 4, starting back in 2001, had always been a rather complicated endeavor. Redesigning them for Blaise 5 proved far more time-consuming than we expected, and we did not expect it to be easy. As described above, the complications emanated from a number of sources. We would not have been doing this if we were not trying to run the whole interview in a self-administered web mode, and that, by itself, upended many of our long-held design assumptions and forced many new design requirements upon us. Blaise 5 itself presented a two-fold problem: we had to discover what options we had in Blaise 5 and learn how to implement them, while at the same time struggling with the fact that the system was new and that we frequently ran into bugs that blocked our progress until they were fixed, or requests for new features or workarounds in Blaise 5 which also took time to sort out. This discussion touched only briefly on the additional complications of having an interviewer-assisted mode (driven by some very different design assumptions) in the same instrument, but the end result was a difficult balancing act. With thousands of interviews now behind us, we can say that we seem to have largely succeeded in building rosters that worked reasonably well for both modes. We already have plans to dive in again next year, though, with the expectation that we will have a more solid foundation for the next effort having been through this process.