Blaise 5 Development at University of Michigan

Max Malhotra and Youhong Liu, University of Michigan Survey Research Center, United States

1. Abstract

We have used Blaise 5 for two survey studies at the Survey Research Center, University of Michigan since two years ago. Blaise 5 and Blaise 4.8 are similar in some ways, but are also different in many other aspects. The two studies we have developed are complex in many areas. One of them is converting a large Blaise 4.8 survey into Blaise 5. The second project is a Mixed Mode Survey. In order to make the surveys flexible, and to present users with friendly interfaces and experiences, we have designed and implemented many different features. We will present how the mixed mode functions are achieved, and how are the challenge requirements programmed. In additional, Blaise source programming, Blaise layout setting, and Resource database development will also be discussed.

Using Blaise 5 for self-administrated surveys posed unique technical challenges due to the complexity of survey design and stringent requirements pertaining to response collection of self-administered web respondents. These requirements mandated the design and implementation of unique back-end APIs that needed to be concise in their delivery so that they could be quickly integrated into a survey using role text to pass parameters from the Blaise 5 data model to the API. This paper looks at how Blaise 5 capabilities and features work with the creation of the aforementioned APIs and we shall show some practical examples of how some of these functions can be best utilized. A technical discussion of issues encountered, what worked well and lessons learned.

In particular, we will discuss:

- Select All, Clear All (Checkbox)
- Mutually Exclusive Response (MER) with Clear All (Checkbox)
- Select Mutually Exclusive Response (MER)
- Other Specify
- Other Specify to clear out textbox
- Single text box with single radio button
- Multiple textboxes with single radio button
- Single textbox with multiple radio buttons
- Multiple dropdown menus with single radio button
- Multiple dropdown menus with multiple radio buttons
- Multiple dropdown menus with multiple radio buttons and with textbox
- Disable Radio Buttons If Mask Textbox Is Zero or Empty
- Disallowing browsers based on browser type and/or browser version

We shall provide a demo of some of the above mentioned custom user interface functionality at the conference.

2. Introduction

The University of Michigan Survey Research Operations began using Blaise 5 at the end of 2014. Over the last two years, we have programmed and implemented two studies using Blaise 5. The first study is a longitudinal study that has been on-going at the University of Michigan since 1968. The Blaise 5 development for this study was implemented in two stages. The first stage was a pilot survey that was conducted using opt-in samples. The second stage was a pilot survey that was conducted among selected study sample.

The second study is also longitudinal, however it began more recently. This study also had two phases. The first phase was a pretest that was conducted at the end of last year. The second phase, which is currently in the field, is the production phase of the study.

The first study is a large survey. It has many fields, loops and preloaded data fields, so the datamodel programming was more challenging. The second study is complicated as well because it is a mixed mode project and the study project staff had more intricate requests with regard to layout and other aspects of the survey.

The Blaise out of box tools, for example, Blaise language, Blaise resource database can help us to achieve most of the functionalities. However, using Blaise 5 for self-administered surveys posed unique technical challenges due to the complexity of survey design and stringent requirements pertaining to response collection of self-administered web respondents. These requirements mandated the design and implementation of unique back-end APIs that needed to be concise in their delivery so that they could be quickly integrated into a survey using role text to pass parameters from the Blaise 5 data model to the API.

As Blaise 4.8 developers with some .net programming experience, there was a big learning curve. We faced many challenges and learned a tremendous amount about programming in Blaise 5. In this paper, we will present some major functionalities that were developed with Blaise tools, as well as API functionality.

3. Mixed Mode Programming

Data collection for one of the studies has two modes: the Self-Administered Questionnaire (SAQ) mode and the Computer-assisted Telephone Interview (CATI) mode conducted by our centralized telephone interviewers. We had several meetings to discuss the best approach to implementing a robust survey that would meet the client's requests, while keeping the process simple. In the end, we decided to use one instrument for both modes and came up with the following implementations:

1. Used a variable called MODE to route surveys differently if SAQ and CATI had different questions. One was for SAQ and two was for CATI. The field value was set by using a parameter that was passed into the survey. We had to pay special attention to the AssignMode parameter. By default, AssignMode was *New*, meaning values would be assigned to the fields in case a new form was created. *Existing* could be used to assign values to fields in existing forms. In our case we used *Always* to assign values to fields in new and existing forms.

2. Used two MODES: SAQ and CATI. MODES in this case is a Blaise 5 key word that is different from the MODE variable created in the survey. It is a new attribute in later releases of Blaise 5. With this new feature added, we were able to specify different attributes between SAQ and CATI easily. In our implementation, *Empty* was allowed in SAQ so Respondents could skip any questions in the survey, But *NoEmpty* is defined in CATI so interviewers could not skip any questions.

3. In the study pretest, Web Browsers were used for both SAQ and CATI interviews. In production, we moved CATI to DEP. It was a smooth transition and we found it was more stable and faster in terms of conducting the survey in DEP and integrating it with other components of the Blaise 5 service.

4. There were four layout sets defined in the survey: Desktop Web, CATI, Medium Mobile Device Web and Small Mobile Device Web. The Desktop Web layout was developed first. It took some effort since we were new to programming in Blaise 5. CATI also took longer to develop because it was significantly different from the Desktop Web layout. When the Desktop Web and CATI layouts were stable, we moved to Mobile Device layout development. The Desktop Web layout

had question table grids and other question elements that would not be displayed properly on small devices. After some investigation, a developer started to work on the mobile device resource set and applied the resource set to a sample survey. After the sample survey was tested by our client, we applied the resource set to our production survey. We found that implementing the survey on different devices in Blaise 5 was very easy by using resource and layout sets.

5. Define two data entry settings, one for each mode. For security reasons, we wanted a short Session Timeout: 10 minutes. This is great approach for the web SAQ mode. But during the pretest in CATI mode, we found a problem on one screen. On that screen, some interviewers spent more than 10 minutes entering data into an open text field. When they moved to the next screen, the "Session Timeout" message came up. At that point, the answers in the open text field were lost. In production, the solution for this problem was to add a special Data Entry Setting that is used for CATI mode only. In that setting, the Timeout is 30 minutes. At same time, the Save function was added on the screen for interviewers to use.

4. Integrating Datamodel and Resource Database Programming

Blaise 5 and Blaise 4.8 are similar, but they are also very different in some aspects. On some of our Blaise 4.8 Window and IS projects, we used expression programming to achieve requests from our clients. For instance, in one of the Blaise IS projects, expression programming in the project's menu editor was utilized. Based on the field name, an external web page was opened on certain screens, so the Respondent could go to that site to take a set of cognitive tests.

But we could not find ways to communicate between Datamodel, Menu, Modlib or other parts of Blaise components. Therefore, there are many functionalities that could not be achieved in the Blaise 4.8 projects. For example, two instruments had to be developed for a SAQ and CATI mixed mode project. This resulted in dual programming - two sets of source code had to be programmed and maintained. With the two instruments approach, the data manipulations were also very challenge.

In Blaise 5, Datamodel and Resource Database programming communications are greatly extended. With careful implementation between these two, we believe we can fulfill almost anything requested by our clients. We elaborate a few examples below.

1. When a case was resumed, it jumped to the beginning of the previously suspended section. Further, if there is a mode switch, the section data should be deleted. One project made this request because some of the data were time related, meaning the follow up questions were related to their root questions and the data needed to be gathered in timely fashion. The deletion after a mode switch was because different set of questions could be asked between SAQ and CATI mode.

   After many tests, below were the sequences we developed:

   - First, two variables in the Datamodel were created: ActiveFieldName and SuspendFieldName. ActiveFieldName was then mapped in the Resource Database.

   - ActiveFieldName was updated in the Master Page's page loading event and then updated on every survey screen. This was done in the Resource database. Since it is a mapped field, the Datamodel recognized the value of the field.

   - SuspendFieldName is updated only when a case is resumed. This was done in the Datamodel.

   - Then in the Datamodel, the resume page was created. It was the first page of the survey. The page will show if SuspendFieldName contains block names, e.g., "Section_A", Section_B", etc. On the page, the resume button was created and actions were attached so that the survey

would jump to the beginning of the section when the button was clicked. This was done in the Datamodel.

- If a mode switch is detected, the section data was deleted.

- At first, the resume buttons were created on the master page of the Resource Database. It was not easy to show and hide, and difficult to add the GotoField event to the buttons. In production, only one button was used; the button itself and its event were moved into the Datamodel by using dynamic enhanced text fills in a text role. This was a significant improvement in terms accuracy and code maintenance.

The list above is the simple description of the function. The actual programming was much more complex. Careful and skillful Datamodel programming is essential. Some intermediate fields were needed. Also *.Keeps* statements are used to ensure that data are properly saved.

2. Determine if a survey is complete. In Blaise 4.8 CATI, we used the following code:

Listing 1. Determining Completion Source Code

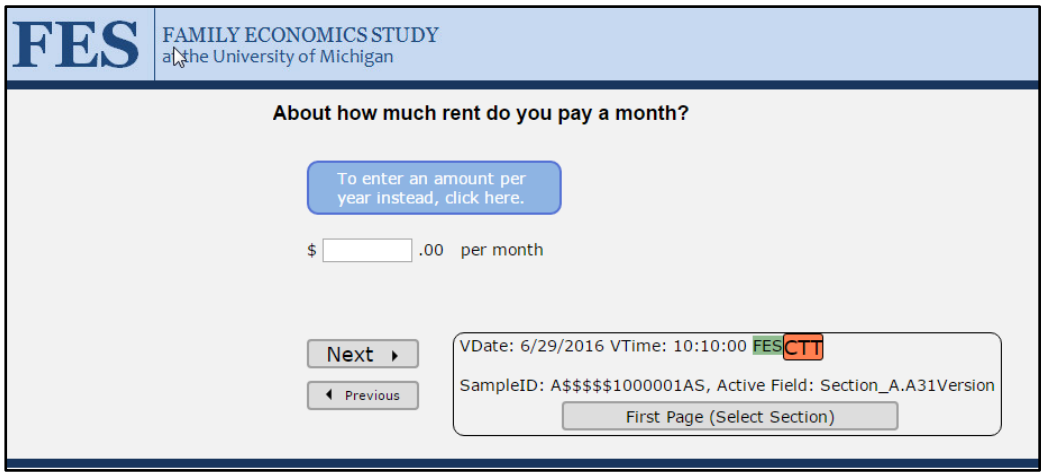
```
Completed.keep
IF Thankyou = Response and Completed=empty THEN
    Completed := Done
ENDIF
```

The above code could also be used in Blaise 5, but there was a problem in the SAQ interview. In the SAQ, we really could not require a Respondent to answer any questions. Therefore, the *Thankyou = Response* condition could not be used and extra steps were required. Our approach was to create a special page just for *ThankYou* question. The *Thankyou* field was mapped in the Resource Database. Then on the master page's "Next" button, Assign *ThankYou = '1'*.

This approach was efficient but had some drawbacks. When the mobile *Thankyou* Master page was defined, we had to add the same code in the page's "Next" button. This could be easily forgotten if we have other Thankyou pages that are used for different devices. We may be able to use the ActiveFieldName described above to program this function as well. The code might be cleaner since everything related is in the Datamodel.

3. Amount/Per Screen with a version button. (See the screen below.)

Figure 1. Amount/Per Screen with Version Button



On this screen, the project staff requested that:

1) The amount per month field is presented to the Respondent by default;
2) A version button is also presented;
3) If Respondent clicks the button, the amount per year is presented;
4) If the button is clicked again, the amount per month is presented again.

We had never done anything like this before, neither in Blaise 4.8 IS nor in any other survey systems. Again, Blaise 5 provided new options. With careful Datamodel and Resource database programming, we were able to create the function.

- First the AmountPer Field Pane template was created. In the template, buttons were created based on the enumeration type of the version button defined in the Datamdoel. The number of buttons, the show/hide, text property and the event behind the buttons were also programed by expression based on the enumeration type.

- Then, on the Datamodel side, the following code was used to make the final show and hide the month and year fields mutually excluded.

Figure 2. Datamodel Source Code

```
GROUP GROUPA31 "About how much rent do you pay ^xA31_1? "
   FIELDS
      A31Version: TVersionButton8

      A31             (A31)
                EntryPre "$"
                EntryDescr".00 per month"
                /  "Rent Amt per month" :  TDollar5

      A31Yr             (A31Yr)
                 EntryPre "$"
                 EntryDescr".00 per year"        /
                "Rent Amt per month" : TDollar5

   RULES
      A31Version.keep
      IF A31Version = EMPTY THEN
         A31Version:=Month_
      ENDIF
      A31Version
      xA31_1:='a month'
      IF A31Version = Year_ THEN
         xA31_1:='a year'
      ENDIF
      IF A31Version = Month_ THEN
         A31
      ELSEIF A31Version = Year_ THEN
         A31YR
      ENDIF
ENDGROUP
```

In this study, they are many Amount/Per screens. Some have more than two versions and others have different texts to be displayed on the buttons. With careful implementation, we were able to use the same template for these similar screens.

5. Designing and Implementing APIs in Blaise 5

We will now look at how Blaise 5 capabilities and features work with the creation of the aforementioned APIs and we highlight some practical examples of how some of these functions can be best utilized, including discussion of challenges encountered.

In particular, we will discuss:

- Select All / Clear All (Checkbox)
- Mutually Exclusive Response (MER) with Clear All (Checkbox)
- Select Mutually Exclusive Response (MER)
- Other Specify
- Other Specify to clear out textbox
- Single text box with single radio button
- Multiple textboxes with single radio button
- Single textbox with multiple radio buttons
- Multiple dropdown menus with single radio button
- Multiple dropdown menus with multiple radio buttons
- Multiple dropdown menus with multiple radio buttons and with textbox
- Disable Radio Buttons If Mask Textbox Is Zero or Empty
- Masks
- Disallowing browsers based on browser type and/or browser version

We shall provide a demo of some of the above mentioned custom user interface functionality at the conference.

5.1 Challenges

Some challenges that we faced were that the current Blaise info-structure did not contain certain project-specific control capabilities that our projects demanded. One example of such a control would be *Select All / Clear All* functionality.

For example if we have a question that asks “In which months did you pay the utility bill last year?” it would be safe to say that a large majority of respondents may pick all twelve months, however we would not want a respondent to have to click on all twelve month checkboxes thus we developed an select all clear all API that allows the respondent to click one button to select or deselect all twelve months and if the respondent manually selects all twelve months the select all checkbox will automatically get selected as well.

Conversely, the respondent can choose to click on the select all button and deselect specific months and once a single month is deselected the API will deselect the select all checkbox as well saving valuable respondent time and providing for a better user experience.

5.2 Implementation

The initial design and construction of each API had to be carefully considered and architected to provide future expandability. The API’s construction utilized a variety of languages such as Blaise, C#, javaScript, and jQuery. However, our goal was to make it very easy to implement for any Blaise programmer and thus from a Blaise programmer perspective -- provided they have our more complex code installed for their specific survey -- the Blaise programmer will simply call the specific API (through a role text) and provide it the variables that are specific for that particular API.

For Example:

- Other Specify

Figure 3. Other Specify Source Code

```
Group GroupA_A07_Q
   Fields
   A_AA_ResnsNoOpt (A_A07_Q)
   "^xA_A07 ^xArmymilitary won't give you the option to reenlist? <I1>(Check all
   that apply)  </I1>"
   /"Army - Reasons they will not let you reenlist"
   RTMod "OtherSpecifyRoleText = Other"
   : Set OF  (_1 "You have a health, disciplinary, or legal problem",
     _2"You have an adverse personnel flag, such as failing to meet physical fitness or weight standards",
     _3 "The ^xArmymilitary is reducing the number of servicemembers in your MOS or eliminating the MOS ",
     _4 "You have a low supervisor recommendation or performance rating",
     _5 "You have reached a Retention Control Point (up-or-out promotion policy)",
     _6 "You are barred from reenlistment",
     Other "Some other reason <I1>(Please briefly describe.)</I1>")

   A_AA_ResnsNoOptSpec     (A_A07_Q)  : TStringOther
ENDGROUP
```

- Select All/Clear All

Figure 4. Select All/Clear All Source Code

```
F7MO           (F7MO)
                "For which months in ^PYear was that?
                <br><br><Instruct>Please select all that apply.</Instruct>"

                "¿Por cuáles meses fue eso en ^PYear?

                <br count=2><tab><Instruct>• ENTER all that apply, For multiple response, use space bar or dash to
                separate responses</Instruct>
                <br count=2><tab><Instruct>• PROBE: </Instruct> ¿Algo más?"
               / "Months in ^P2Year Paid Child Care"
               RTMod "SELECTALLCLEARALL = All"
               :

                TMoStringSet
```

5.3 Examples

- Select All, Clear All (Checkbox)

  SPEC: 'Select All' (e.g., All 12 months) as a check box at the top
  When checked, all check box items are checked
  When unchecked, all check box items are unchecked
  When all Items are checked the 'Select All' gets automatically checked
  When all Items are checked, including 'Select All' and one item is unchecked, then the 'Select All' checkbox is automatically unchecked

Screen Capture:

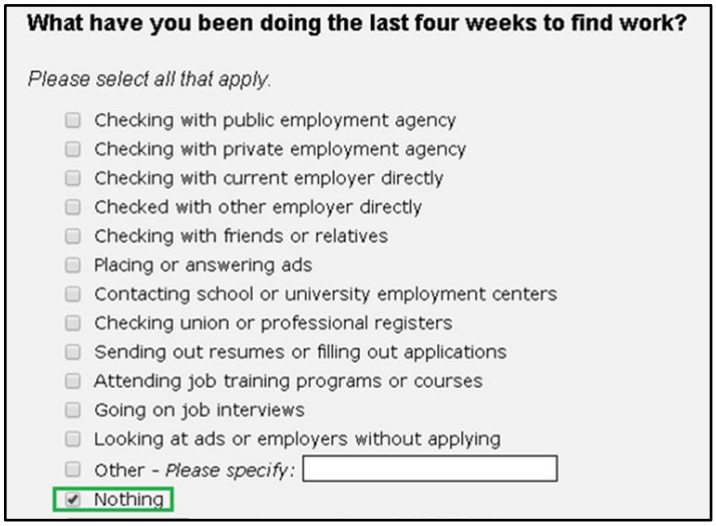
Figure 5. Select All, Clear All Checkbox

- Mutually Exclusive Response (MER) with Clear All (Checkbox)

SPEC: 'Select None' (e.g., Nothing) as a check box at the bottom
When checked all check box items are uncheck
When checked if 'Other Specify' textbox exists, the textbox is cleared out.
When ANY checkbox item other than 'None' is checked, then the 'None' item is unchecked.

Screen Capture:

Figure 6. MER with Clear All Checkbox



- Select Mutually Exclusive Response (MER)

Depending on how this API is used, the API can function as a separate API that is capable of making individual checkbox items mutually exclusive as well.

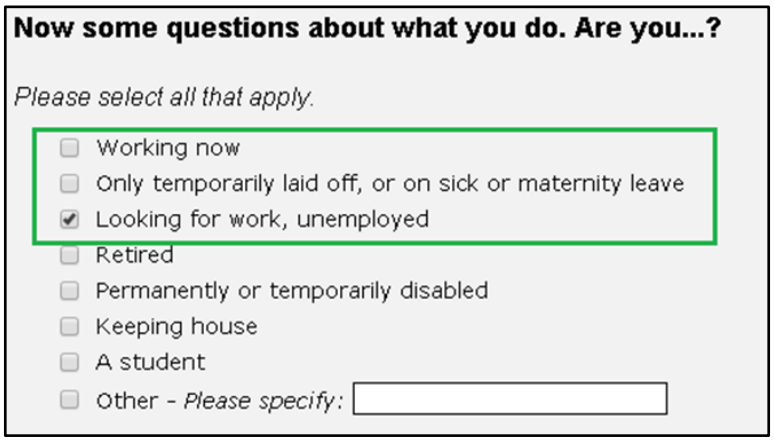
SPEC (EXAMPLE):

Response options one, two and three are mutually exclusive and all other items are independent of this API.

This means if response option one is selected and then response option two is selected, then response option one will be deselected. Only one of the three allowed response options can be selected at a time. The other two cannot be selected at the same time. All the following response options are not impacted by this API.

Screen Capture:

Figure 7. MER

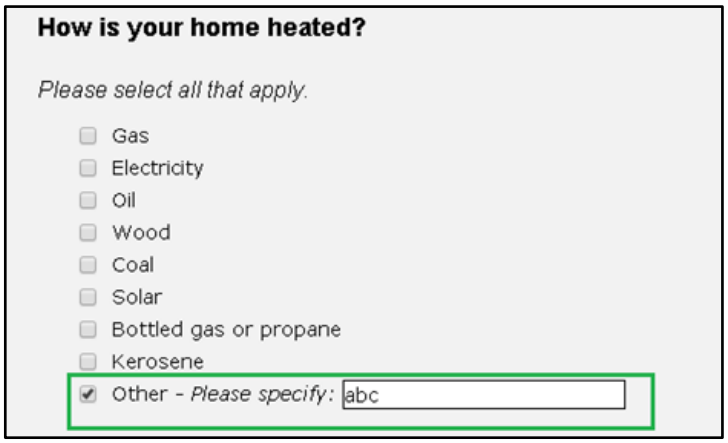


- Other Specify

  a. CHECKBOX: When the respondent clicks on or starts to type inside the 'Other Specify' textbox a checkbox shall automatically be assigned to that item.

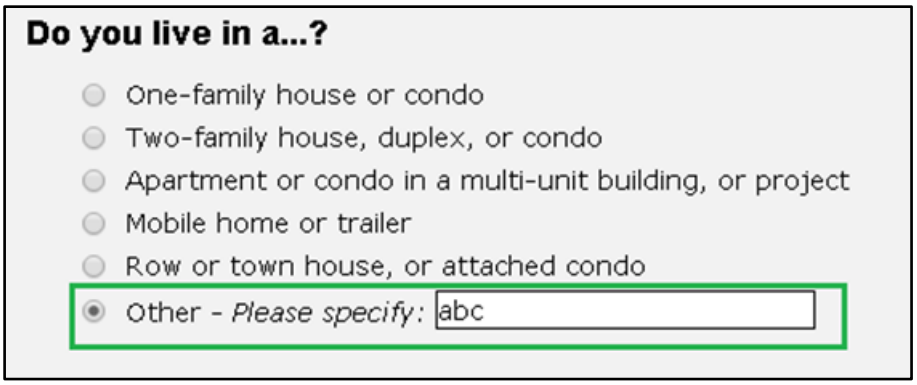
Screen Capture:

Figure 8. Other Specify



  b. RADIO BUTTON: When respondent clicks on or starts to type inside the 'Other Specify' textbox a selected radio button shall automatically be assigned to that item.

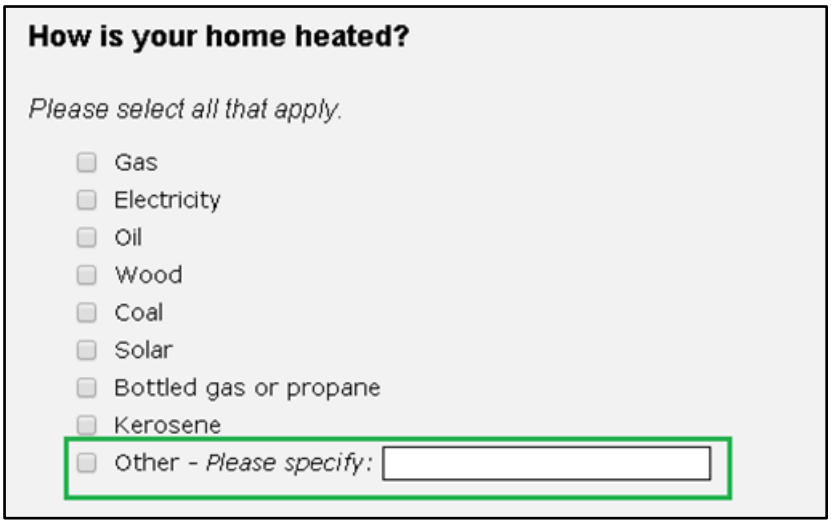
Figure 9. Radio Button



- Other Specify to clear out textbox

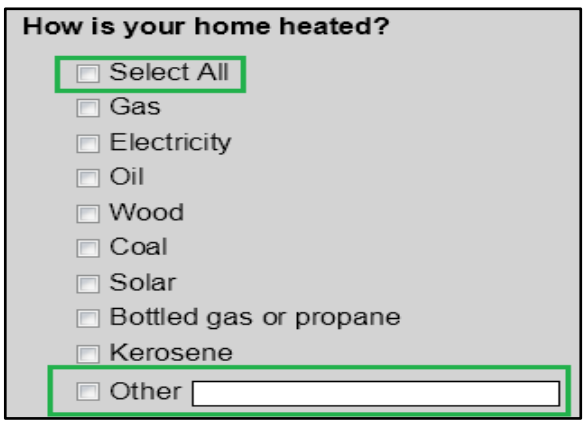
When checkbox is unchecked any entry inside the textbox is cleared.

Figure 10. Other Specify to Clear Out Textbox



c. If 'Select All' (checkbox) option is present then and 'Select All' is deselected, then the 'Other Specify' textbox shall be cleared out as well.

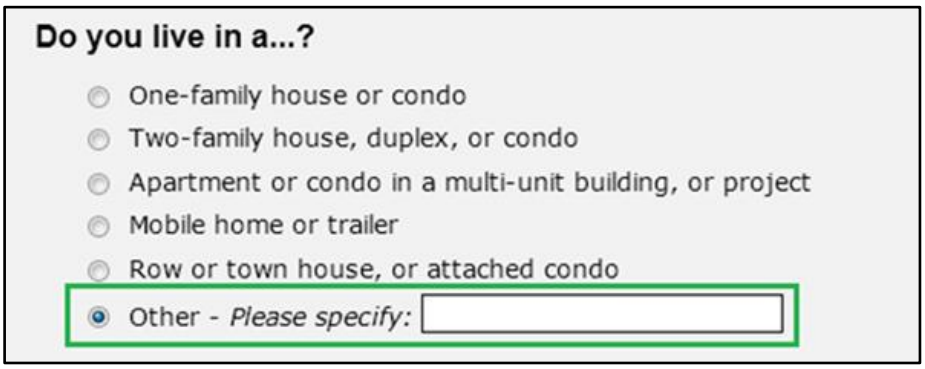
Figure 11. Select All/Other Specify



d. RADIO BUTTON: If respondents type inside the 'Other Specify' textbox and then decide to switch to a different radio button, the text inside the 'Other Specify' is cleared out.

Screen Capture:

Figure 12. Radio Button/Other Specify



e. The following screen shows same process for the "Other – *Please specify*:" function.

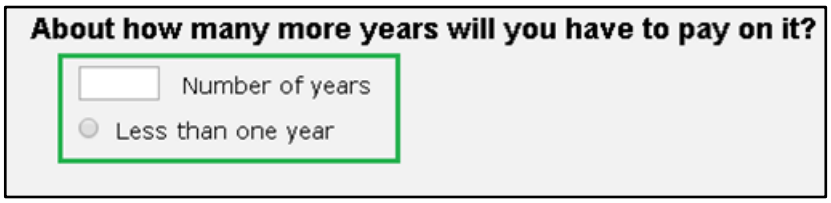
Figure 13. Radio Button/Other Specify

- Single text box with single radio button

SPEC: If the respondent types inside the textbox and then decides to click on the radio button, then the text box shall be cleared out. If respondent selects the radio button and then decides to click or start typing inside the textbox, then the radio button shall be cleared out.

Screen Capture:

Figure 14. Single Text Box, Single Radio Button

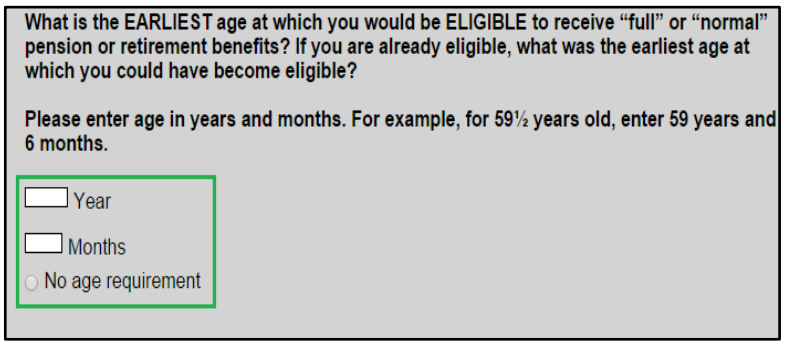


- Multiple textboxes with single radio button

SPEC: If the respondent types inside any textbox and then decides to click on the radio button, then ALL the text boxes shall be cleared out. If the respondent selects the radio button and then decides to click or start typing inside any textbox, then the radio button shall be cleared out.

Screen Capture:

Figure 15. Multiple Text Boxes, Single Radio Button

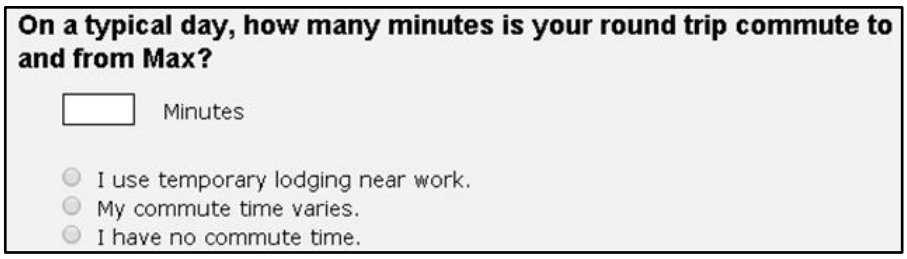


- Single textbox with multiple radio buttons

SPEC: If the respondent types inside the textbox and then decides to click on ANY of the radio buttons, then the text box shall be cleared out.
If the respondent selects ANY of the radio buttons and then decides to click or start typing inside the textbox, then the radio button shall be cleared out.

Screen Capture:

Figure 16. Single Text Box, Multiple Radio Buttons

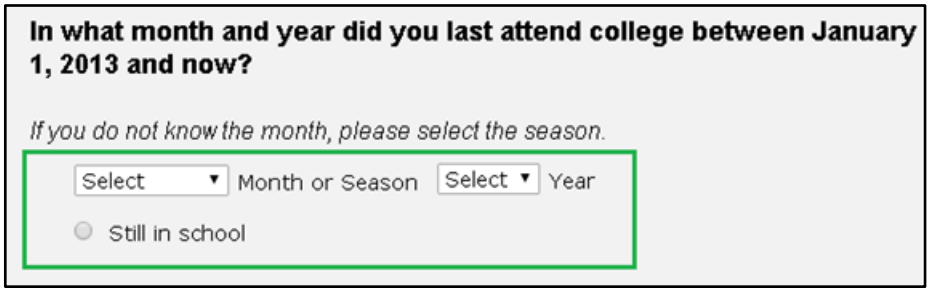


- Multiple dropdown menus with single radio button

SPEC: If the respondent selects ANY dropdown menu and then decides to click on the radio button, then ALL the dropdown menus shall be cleared out. If the respondent selects ANY dropdown menu then the radio button shall be cleared out.

Screen Capture:

Figure 17. Multiple Dropdown Menus, Single Radio Button



- Multiple dropdown menus with multiple radio buttons

  SPEC: If the respondent selects ANY dropdown menu and then decides to click on the radio button, then ALL the dropdown menus shall be cleared out.
  If the respondent selects ANY dropdown menu, then ANY selected radio button shall be cleared out.

  Screen Capture:

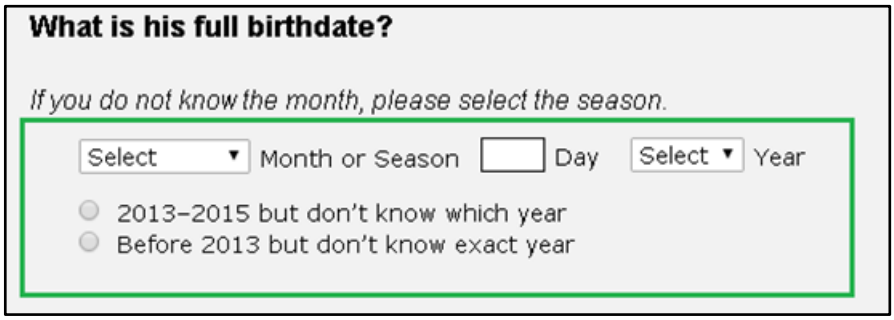
Figure 18. Multiple Dropdown Menus, Multiple Radio Buttons

- Multiple dropdown menus with multiple radio buttons and with textbox

  SPEC: If the respondent selects ANY dropdown menu or textbox and then decides to click on the radio button, then ALL the dropdown menus and textboxes shall be cleared out.
  If the respondent selects ANY dropdown menu or textbox, then ANY selected radio button shall be cleared out.

  Screen Capture:

Figure 19. Multiple Dropdown Menus, Multiple Radio Buttons with Text Box



- Disable Radio Buttons If Mask Textbox Is Zero or Empty

  SPEC: Radio Button: If the respondent attempts to click on the radio button when the masked textbox is equal to zero or empty, the radio button shall automatically be unchecked and disabled.
  Other Specify: If the respondent attempts to click on the 'Other Specify' textbox, the radio button shall be cleared out if mask textbox is zero or empty.
  If the respondent attempts to type inside the 'Other Specify', the text field shall be cleared out and grayed out.
  Number Text Box (Numeric Mask Text Box): Once the respondent clicks on the box or starts typing or makes a change, the radio buttons and the 'Other Specify' textbox shall be disabled and grayed out if the value is empty or outside of the bounds for the range set for that field. If the value is inside the range set for that field, then the radio buttons and 'Other Specify' textbox shall be enabled and the box shall change from gray to white.

Screen Capture: On page load:

Figure 20. Empty Mask Text Box, Disabled Radio Buttons

After the value (inside correct range) is entered into mask textbox:

Figure 21. Mask Text Box with Entered Value and Radio Buttons

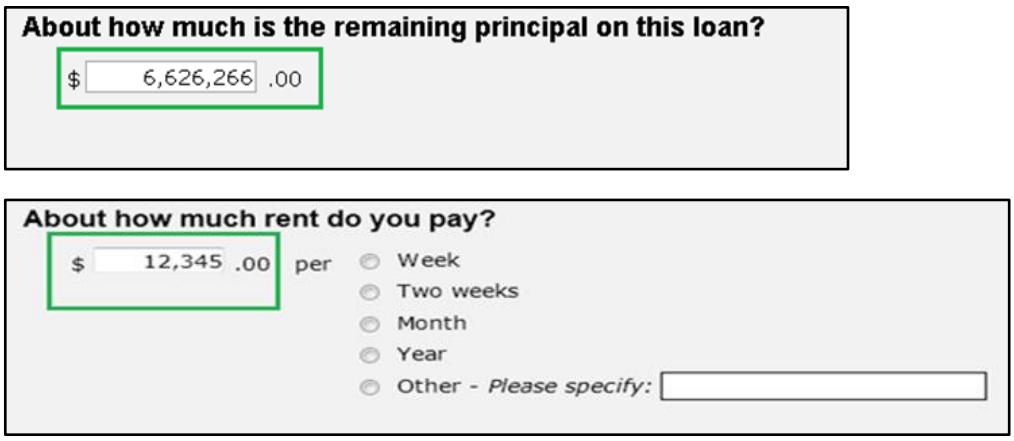
- Masks

Masks can be used in multiple ways:

1. Inserts appropriate group separators every 3 digits on a field (e.g. , a comma)
2. Ability to display (in the entry box) the prefix to a number, such as the currency symbol
3. Limits input between the maximum, minimum, and number of digits to the right of the decimal, according to the field definition
4. Empty values are allowed
5. "0" may/may not be allowed, depending upon the range of the field
6. Decimal places are not allowed if not specified in the field
7. The resulting custom control's width is automatically adjusted to the expected width of the entry field.
8. Values less than the minimum or more than the maximum are not allowed
9. Single key presses that make an invalid entry are deleted.
10. The mask action will truncate input that exceeds the width of entry (e.g., hold a key to allow it to repeat, and then release it: "9999999" may truncate to "999")
11. Selecting all the input and overwriting it with invalid data will cause the old input to reappear. For example, "230" is selected and overwritten with "999999999" (a value larger than the field can hold) the result will be 230.

Screen Capture:

Figure 22. Masks



6. Conclusion

In this paper, we only described a few functionalities. There were many other functions developed by using Blaise 5's new and updated features including APIs. We had a steep learning curve in developing some of the functions. However, once they were properly designed and implemented they have become an integral part of our project design workflow. These provide numerous benefits in terms of more accurate data collection, and likely less respondent attrition due to preventing frustration, and providing a more cohesive respondent experience. In the future we intend to expand our library and continue to grow as project needs warrant additional functionality.

In conclusion, we have gained significant knowledge in Blaise 5 in the past two years. We believe that Blaise 5 is a full-fledged survey software system that helped us to build robust surveys.

7. References

1. Blaise 5 Help Manual

2. Stackoverflow.com/ and other Internet resources