Design Considerations for Web and CAPI Multimode Using Blaise 5

Todd Flannery and David Simpson, Westat

Presenter: Todd Flannery

1. Introduction

Differences in modes like CAPI and CAWI present some unique challenges related to data collection and storage for a single instrument. For example, the web may require token authentication that is not needed in a secure disconnected tablet environment. In Blaise 5, we have tools that allow us to present question text and create unique routes that are based on either the mode or are perhaps device specific for the web. This paper describes several of these methods used to allow synchronized code updates between multiple modes by using prepared directives along with other specialized techniques to develop the code base.

For some projects, both the web and disconnected CAPI are used in order to collect instrument data from sampled respondents. To accomplish this, we have designed and developed an instrument in Blaise 5 that allows us to handle difference in the modes. In order to maintain comparability of the data, we need to ensure that the text of the questions and responses are consistent, but each mode has some unique considerations that also need to be addressed.

2. Some Initial Considerations

Time is not always on your side. We made the decision early in the design stage that our modes were different enough to forego trying to align the data model structures perfectly. To this end, the data and paradata can be separated into web or CAPI when they are being collected. Although this approach results in a greater amount of work in terms of the data alignment on the back end, it is less constraining on the data model design and development.

Web design requires more focus on accessibility controls, security, and maintaining mode-specific templates and layouts that can be used on a variety of devices and browsers. There are other CAPI-specific controls like touchscreen use, electronic signatures, or text-to-speech that may require specialized code like actions setups that may not be available via a client-server configuration. Additionally, since a disconnected CAPI instrument uses the Windows Data Entry Program, consideration needs to be given to developing layouts for a non-browser–based instrument. Although many differences in template design by mode can be effectively seen in the layout design, each mode (and browser) should be tested via data entry to detect errors in the layouts. Several elements of web versus CAPI design are depicted in Figure 1.

3. Code Methods—Use of Conditional Defines

Conditional defines (Figure 2) can be used to maintain separate project settings while having each project use the same files, and we can then allow the same text, fields, rules, or other Blaise controls to be shared among the projects. If we align the projects to specific modes, we can then update things like text and guarantee comparability over multiple modes. For our study, we have a mode (or project) for CAPI and several modes aligned with the different web servers (development, testing, production, etc.) When we build the BDIX files and associated objects like SQL tables or SQLite database files, they are associated with the mode that is defined for the specific project. This allows us to open the project to only view the routes, definitions, and layouts for the defined mode. For our project, the focus on maintaining consistent text translates into keeping the question texts and response texts outside of these conditions, whereas the mode-specific text roles, procedures, blocks, fields, routes, or layouts may be exclusive to the mode in which the build is occurring (Figure 3).

Figure 1. Web Design and CAPI Design Elements

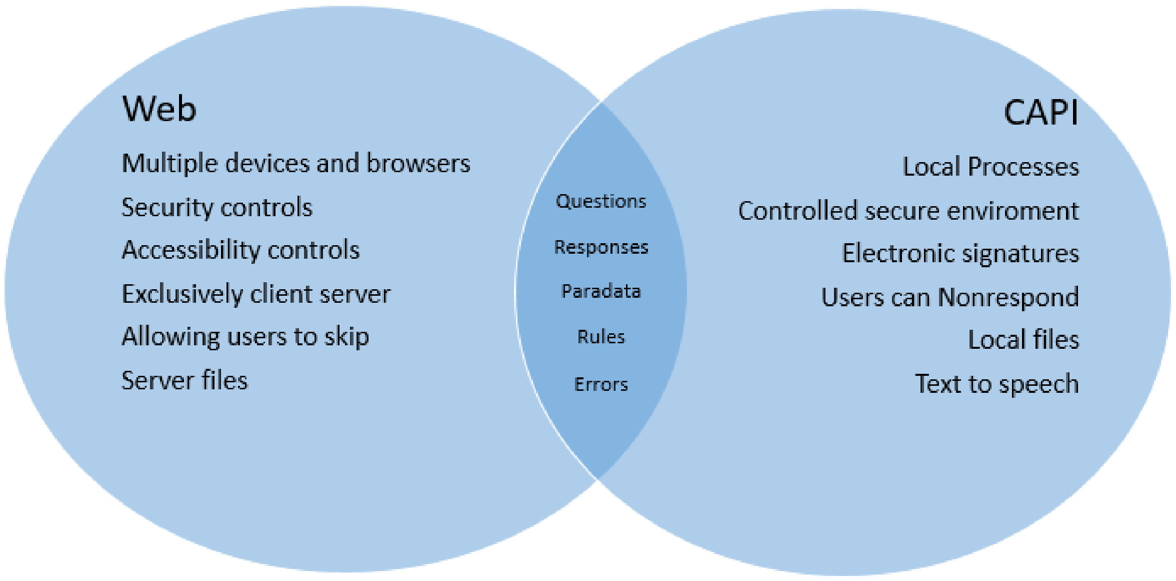
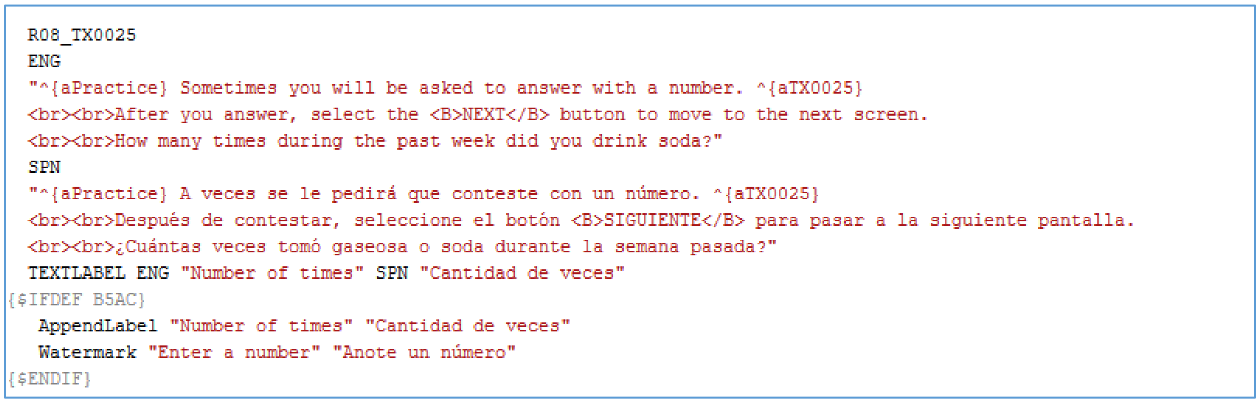
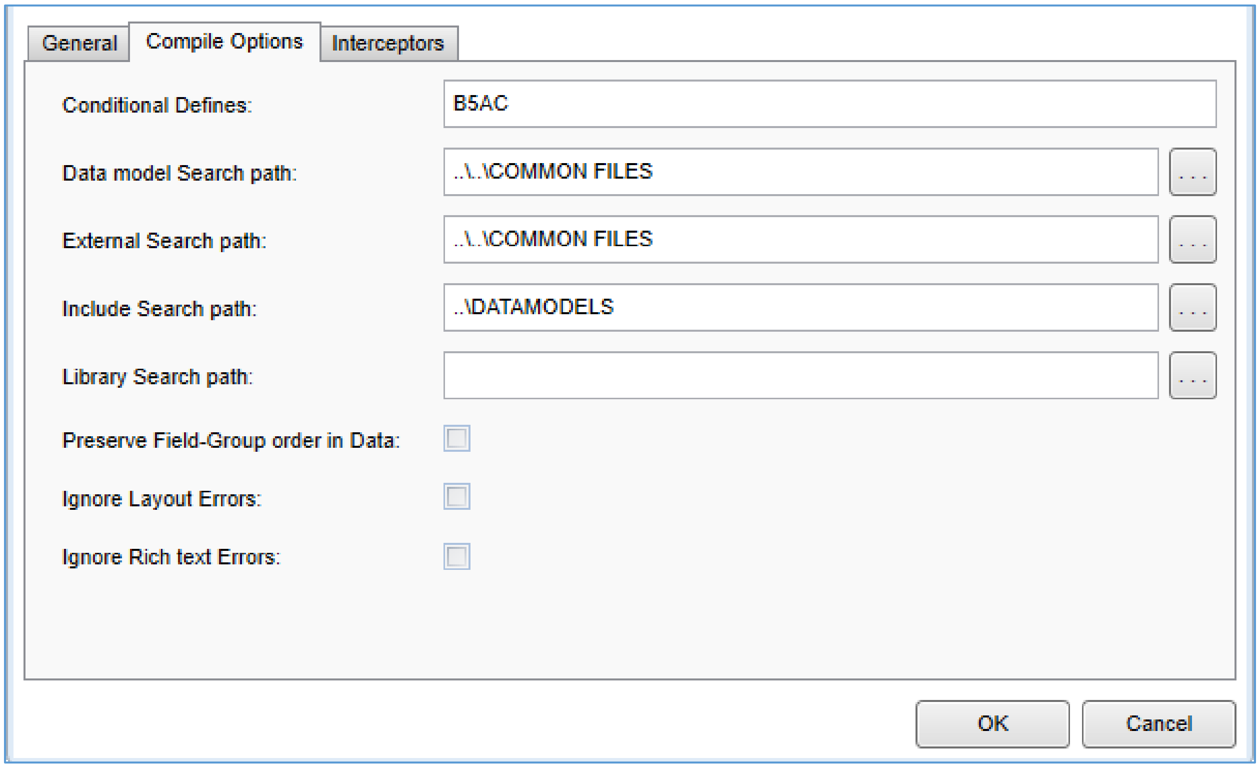


Figure 2. Use of Conditional Directive for a Non-Web Screen



```
  R08_TX0025
  ENG
  "^{aPractice} Sometimes you will be asked to answer with a number. ^{aTX0025}
  <br><br>After you answer, select the <B>NEXT</B> button to move to the next screen.
  <br><br>How many times during the past week did you drink soda?"
  SPN
  "^{aPractice} A veces se le pedirá que conteste con un número. ^{aTX0025}
  <br><br>Después de contestar, seleccione el botón <B>SIGUIENTE</B> para pasar a la siguiente pantalla.
  <br><br>¿Cuántas veces tomó gaseosa o soda durante la semana pasada?"
  TEXTLABEL ENG "Number of times" SPN "Cantidad de veces"
{$IFDEF B5AC}
   AppendLabel "Number of times" "Cantidad de veces"
   Watermark "Enter a number" "Anote un número"
{$ENDIF}
```

Figure 3. Definition for the Non-Web Build Mode of a Project



Conditional Defines:	B5AC
Data model Search path:	..\..\COMMON FILES
External Search path:	..\..\COMMON FILES
Include Search path:	..\DATAMODELS
Library Search path:	
Preserve Field-Group order in Data:	☐
Ignore Layout Errors:	☐
Ignore Rich text Errors:	☐

4. Web-Specific Considerations

For the web, some additional controls are required to be able to prevent security breaches into the study data or maintain compliance with accessibility standards (508, WCAG). Token authentication involves passing an expiring unique identifier (i.e., the token), like a GUID, into a Blaise instrument to be verified against an external data source before it expires. This ensures that access to the study data is only allowed from a known source, such as a landing page for a website. Since the source of the Blaise instrument is completely controlled in a disconnected mode, there is no requirement to token authenticate for CAPI, so we can exclude that section of code based on the mode. Blaise 5 help provides documentation for accessibility requirements settings and has the example “Visually Impaired” to demonstrate features associated with accessibility. Depending on project requirements, these controls may be mode-specific, so we can use the conditional defines to define our metadata based on these mode requirements.

5. CAPI-Specific Considerations

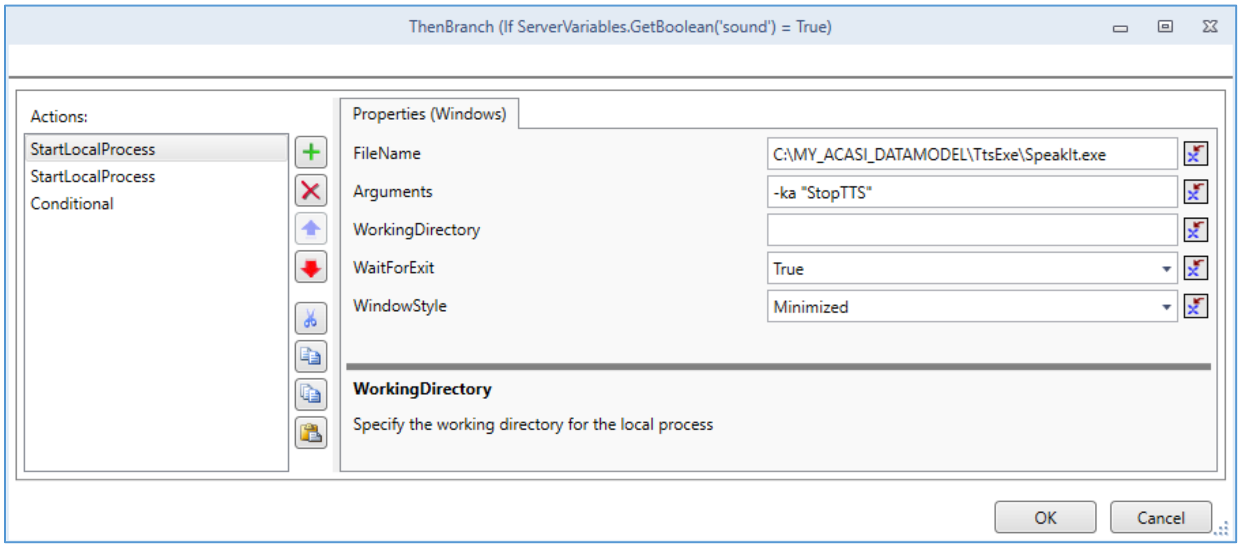
5.1 Action Setups

Blaise 5 allows use of one action setup per project to shell to either Manipula or non-Blaise processing, like an application to collect e-signatures. This method has been very useful in our project to allow some nonstandard Blaise behaviors like looping back to an earlier field on the route without first triggering an error message. If your user is a respondent, behaviors like these can be simpler to administer than training the user to react to error messaging. Since Blaise limits these action setups to thick client mode, some actions that are desired for the web need to use alternative methods to do this additional processing.

5.2 StartLocalProcess

We find the StartLocalProcess method via expressions in the resource database to be very useful in disconnected mode, as an alternative to using an action setup because the StartLocalProcess method, unlike Action Setup, does not require a page refresh (Figure 4). The action can call an external program to perform multiple actions. This in turn allows some processing to be done outside of conventional Blaise behaviors while a user may have scrolled down to the bottom of a page (a page refresh would by default show the top of the page).

Figure 4. Using StartLocalProcess in the Events Editor Instead of Action Setup to Call an External Process and Send Some Parameters



FileName: C:\MY_ACASI_DATAMODEL\TtsExe\SpeakIt.exe
Arguments: -ka "StopTTS"
WorkingDirectory:
WaitForExit: True
WindowStyle: Minimized

WorkingDirectory
Specify the working directory for the local process

6. Both Web and CAPI Considerations

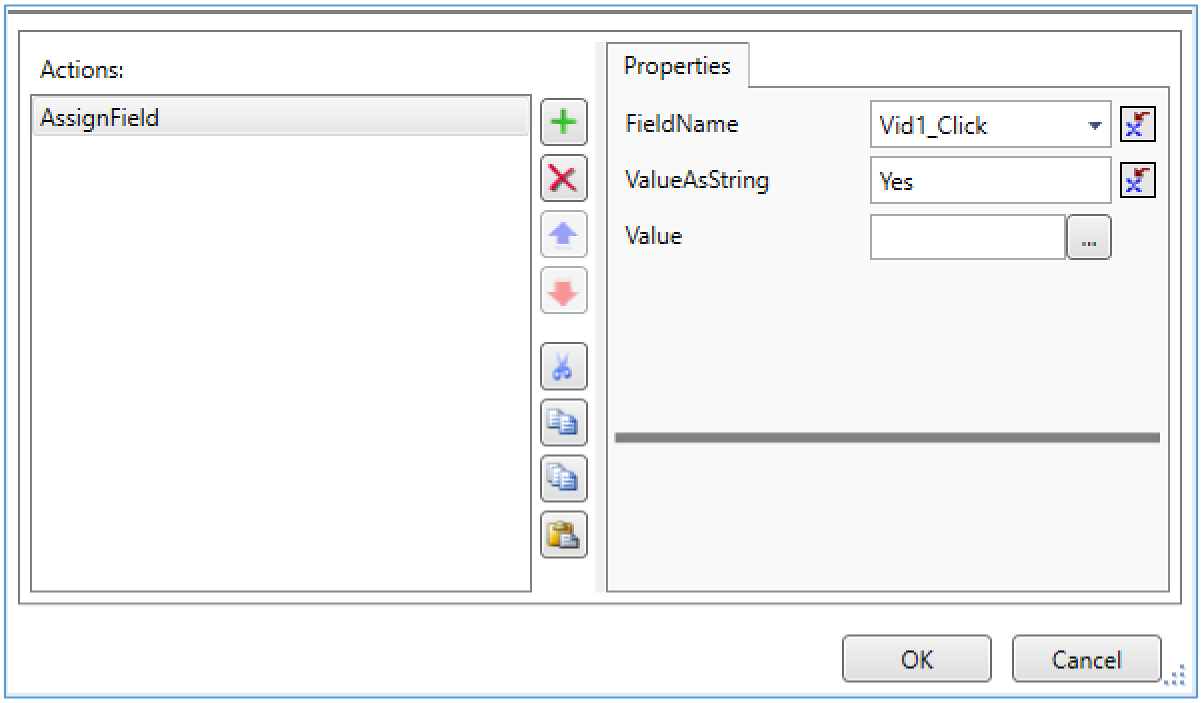
6.1 Texts

Since the content of the questions and responses is critical to the data quality, we need to ensure that any edits to these texts are simultaneously applied to both modes. The easiest way to satisfy this requirement is to have a single source for both modes. So, aside from mode-specific text (e.g., CAPI interviewer instructions), we share text across all modes for instrument questions and responses. Additional text roles that do not involve question or response text, like alt text for accessibility via the web or CAPI-interviewer error messages for CAPI mode, can be specific to the mode at build time.

6.2 Using Expressions and Assignments in the BLRD

To be able to capture user behaviors that trigger data assignments, we can use expressions or assignments in the events editor (Figure 5). This allows us to confirm whether an image or video was clicked on and can determine routes or error messages.

Figure 5. Assigning to Field Reference Based on a Click Event



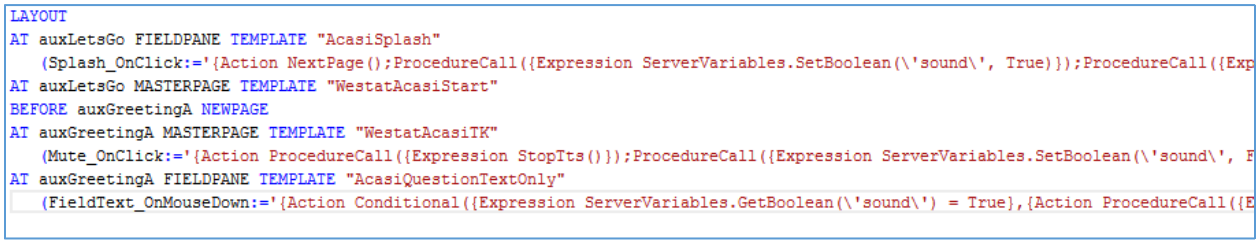
AssignField

FieldName	Vid1_Click
ValueAsString	Yes
Value	

6.3 Use of Server Variables

Server variables can also be useful to allow assignments based on user behaviors, like clicking on a "Next" button instead of responding to something on the page or initiation of some ACASI events. These values can also be passed to procedures in an action setup for thick client configurations (Figure 6).

Figure 6. Using Server Variables to Store Boolean Values Used in Expressions.



```
LAYOUT
AT auxLetsGo FIELDPANE TEMPLATE "AcasiSplash"
   (Splash_OnClick:='{Action NextPage();ProcedureCall({Expression ServerVariables.SetBoolean(\'sound\', True)});ProcedureCall({Exp
AT auxLetsGo MASTERPAGE TEMPLATE "WestatAcasiStart"
BEFORE auxGreetingA NEWPAGE
AT auxGreetingA MASTERPAGE TEMPLATE "WestatAcasiTK"
   (Mute_OnClick:='{Action ProcedureCall({Expression StopTts()});ProcedureCall({Expression ServerVariables.SetBoolean(\'sound\', F
AT auxGreetingA FIELDPANE TEMPLATE "AcasiQuestionTextOnly"
   (FieldText_OnMouseDown:='{Action Conditional({Expression ServerVariables.GetBoolean(\'sound\') = True},{Action ProcedureCall({E
```

7. Data ALIGNMENT and Handling Field Updates

The desire to design and develop separate data structures that use the same texts leads to some additional complexities when transferring data between modes or merging data for delivery. For example, in order to maximize the response rate, some studies will allow a web user to skip past a field without responding, whereas CAPI mode requires the respondent to enter a nonresponse value of "don't know" or "refused" before progressing in the route. Since these definitions may contain different values, you must consider how to store, transfer, or deliver the data.

8. Results

Overall, results have been successful in the approach to use a single code base for text and have the metadata structures and layouts designed and developed based on the mode. Additional time is required to process and analyze test results, since the data are dependent on the mode. The experience has enriched our knowledge base of each mode and will inform future decisions regarding how to adapt to multimode requirements to more efficiently combine or separate the design and development for each mode.

9. Limitations of the Approach

One of the benefits or drawbacks of the approach is that conditional directives can appear anywhere in the code base, such as field definitions, rules, or layouts. So maintaining or discerning where modes differ between the built data models can be complex without strong source code management. Additionally, since the conditions themselves drive the build process but are not part of it, making sure that all of the "{$IFDEF}", "{$ELSE}", and "{$ENDIF}" statements can be burdensome. To assist with this task, it is helpful to know that an enhanced search of the number of $ENDIFs necessarily needs to be the sum of $IFDEFs and $IFNDEFs. To be sure, trying to track down missing conditional define statements can be difficult.

Given ample time for development and design, a single data model that allows data in either mode may be optimal, since the approach for using separate data structures by mode also requires significant commitment to merging data as a back-end task. So, given that design elements like layouts or web-based controls will need to be treated separately, consideration needs to be given as to whether the additional task of maintaining separate data is desired as well, especially if your project is intended to be fully multimode, such as when a respondent is given the opportunity to start in one mode, save and exit, and then finish in another mode.

10. Where to Go from Here

- Greater specificity of controls for the modes: Refining how to apply the conditional directives involves some trial and error to achieve optimal behavior and storage of the data in the data models. As we continue to learn more about how the design of web instruments improves our data collection capabilities, we can work to adapt how each mode can be made more efficient.
- As time allows, align the data. Ideally, one database shared among modes would allow for real-time multimode sharing of the data. Achieving this goal also eliminates the need for back-end processing to align the data, as well as the need to do multi-instrument synchronized releases. An intermediate step may be to have a server-based process to copy data from mode to mode, but this requires some lag time to handle the copying.
- Refinement of the processing: We apply many concepts from the exploration of multimode instrument development to include actions that occur outside of Blaise, enabling greater flexibility in our handling of on-screen behaviors and actions. Additionally, we can continue to apply some of the techniques like the use of server variables or adding events to enhance web processing to allow greater ease of use for users.
- We can continue to refine template design to allow the use of parameters in the templates themselves in order to have a consistent appearance across modes wherein a single design change can be applied to templates that are used in either mode.

11. Conclusions

Design and development for multimode instrumentation requires significant planning in determining which approach regarding database structure, code base, resources, and layouts is exclusive to each mode. Given substantial time to plan this work, a single database and codebase could be achievable, but complex studies rarely provide for this benefit. To be able to collect high-quality data via the web and disconnected modes, the approach to utilize these techniques of retaining a single code base for text and allowing for differences in data structure and layouts has proven beneficial in circumstances where complete harmonization of the data between modes is not possible in a single structure.