Jumping around in Blaise IS

Maurice Martens, CentERdata

1. Introduction

In recent years CentERdata has developed several Life History Calendars (LHC). A LHC is an interactive scaled calendar representation on which annotated life events are visually marked by dates or years of occurrences. For CAPI projects these calendars have been developed using the Blaise API combined with a Visual Basic (VB) shell in order to represent the calendar interface. For a web version we scripted it using PHP and JavaScript, independent of Blaise.

Recently CentERdata was asked to support a study by providing such a LHC. This study was conducted by doing two hour face to face interviews with respondents with a criminal background. As the development team of CentERdata, we considered reusing the VB version, but since the laptops that were used for conducting the interviews had an internet connection available, and since preventing any loss of data was very important for this hard to reach group, we decided to try to implement a web survey that runs the CAPI over the web using Blaise IS.

The questionnaire with a LHC has some specific properties. It should always be possible to jump directly back to earlier questions. The calendar is a representation of events in a person's life, it should always be visible. The questionnaire consists of a series of large loops over events that persons would have had in their lives. This generated a large number of fields, pages and tables, which caused the questionnaire to slow down the questionnaire immensely. To solve this problem and to speed the questionnaire up we implemented some locks in the code to make sure the forward route checking would not load the complete questionnaire in memory.

Initially we tried to generate the html code of the calendar in Blaise, but due to memory constraints we ended up creating an independent service (using PHP) to generate the html code that displayed the calendar which was loaded inline using asynchronous calls. JavaScript would then trigger client side visualizations and actions performed on the presented calendar.

Based on these experiences a more generic tool was developed that can be included in any Blaise IS questionnaire. This tool builds a history of the questionnaire fields visited. It allows you to jump freely through Blaise IS questionnaires. For each field a status can be set and remarks can be made, giving us the basic features of an online testing app for Blaise IS questionnaires.

2. Online life history calendar

A Life History Calendar refers to a survey method in which events of a respondent's life are visualized in a table structure. CentERdata was asked to develop a Life History Calendar for a study in the Netherlands that wanted to have a retrospective interview with people who have a criminal record.

We assumed these (former) criminals were reluctant to participate, which raised the value of the interviews. Therefore we had to make sure a completed interview could never be lost so we avoided storing data on the laptops during the interviewers. We decided to store the answers on an external server over a secured line. After some investigations we decided that mobile internet coverage was good enough in the Netherlands to support this technique. In this decision it was unfortunately overlooked that some institutionalized respondents are not allowed to have internet access at all.

We grouped life events into thematic groups: partners, children, accommodations, jobs, prison, health and other. The interview routing would by default walk through the questionnaire sequentially, group after group. E.g. the first group asks about children- for each of the children when they were born, what their names are, if they are still alive, if not, when they died. When all children are discussed the questionnaire continues looping through all notable romantic relations the respondent has had and so on. All the known events are represented in the calendar by separated lines. The vertical axis of the calendar showed the lifespan of the interviewee.

The calendar does not only give a graphical overview of life events, but can also be used to navigate through the questionnaire. The idea is that respondents, triggered by the visual representation, mentally link life events to each other. Rather than exactly knowing what year an event took place a respondent is more likely to have remembered that an event occurred at the same time as or earlier or later then another event. This makes it more likely that respondents will find an earlier answer was not correct. Using the links provided by the calendar it is possible to jump directly back to these questions.

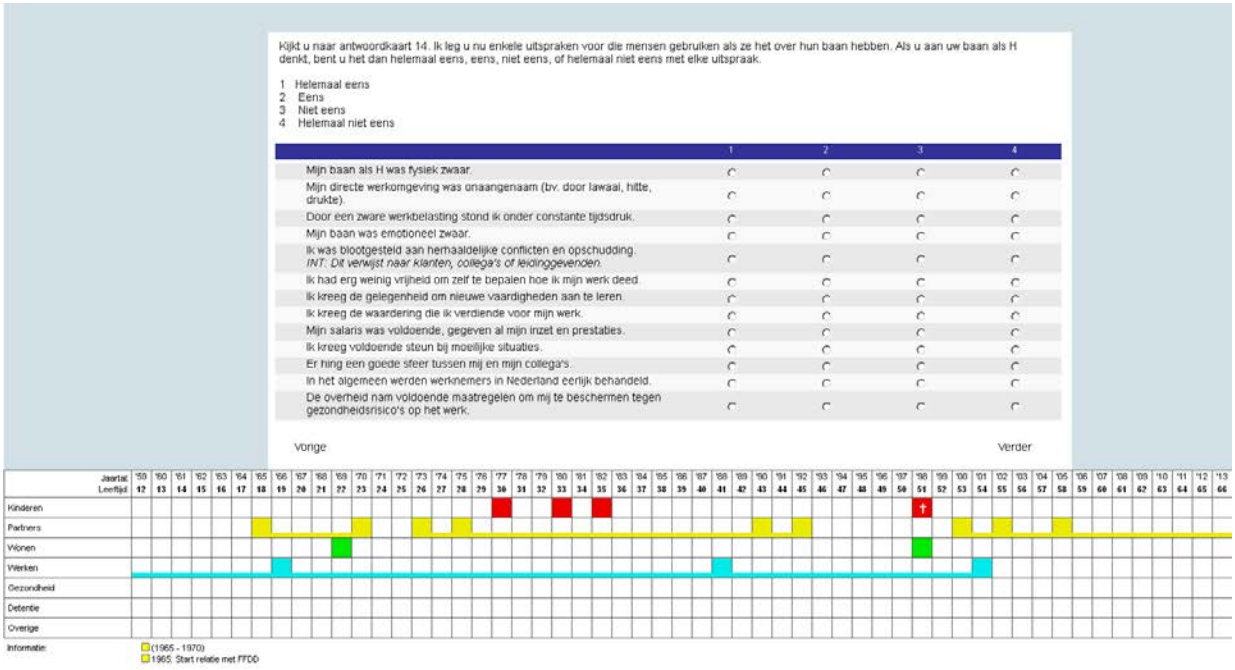

Figure 1: Screenshot of the online LHC

The colored blocks in the calendar represent certain events. When such a block is hovered with the mouse the bottom part of the calendar ('Information' field) displays detailed information on these hovered events. When a colored block in the calendar is clicked the displayed details of the events are locked into place and continue to stay visible. The detailed information contains links that will navigate the user (back) to the questions that set the values for the event.

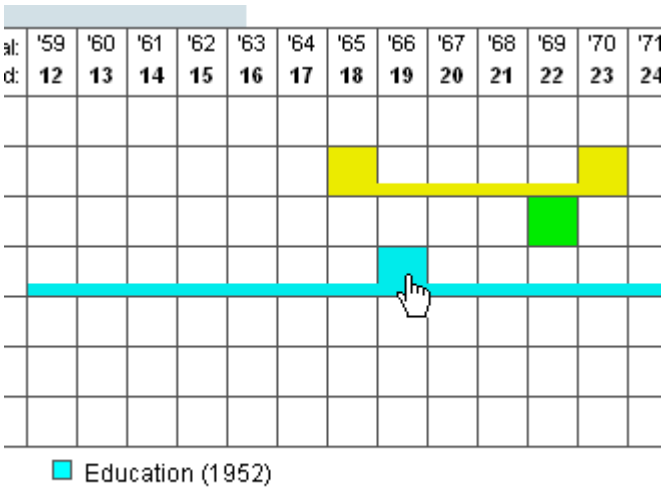

Figure 2: Selecting a life event

3. Calling an external service

In Blaise a string is written that contains a JSON object that describes how the calendar should be displayed initially:

```
CALENDAR:=
'<script language="JavaScript">
   calendar.push ({
      "Initialize":{
      "Key1":'+STR(nohouse)+',
      "Key2":'+STR(nomem)+',
      "StartYear":' + STR(age+YEAR(SYSDATE)) + ',
      "EndYear":'+STR(YEAR(SYSDATE))+',
      "Sections":{
         "1":{"Label":"Children","Link":"Sec_RC.RC001_strkid"},
         "2":{"Label":"Partners","Link":"Sec_RP.RP001_prtstart"},
         "3":{"Label":"Living","Link":"Sec_AC.AC001_acstrt"},
         "4":{"Label":"Jobs","Link":"Sec_RE.RE001_whstart"},
         "5":{"Label":"Health","Link":"Sec_HS.HS001_HSStart"},
         "6":{"Label":"Detention","Link":"Sec_PR.PR001_PRStart"},
         "7":{"Label":"Other","Link":"Sec_OT.OT001_OTStart"}
      }
   }});
</script>'
```

Using an Initialize procedure this string is loaded into a variable and can be called in any question text as a fill. This will set the global properties of the calendar. For updating the calendar we implemented another procedure, SaveEvent, which is called with 10 parameters:

- `piSectionIndex` : the section in which the event is stored
- `piRangeIndex`: the range number within the section
- `piEventIndex`: the event number within a range
- `piLabel`: the label that will show on mouse over on the event

- `piRangeLabel`: the label that will show on mouse over on the range
- `piSymbol`: a symbol that will show in the event tab
- `piYear`: the year in which the event took place
- `piShowRange`: whether the range will be visible
- `piEndRange`: whether the range has an ending
- `piLink`: the full path and name of the question

The procedure will set a call to a function of a calendar class named push that forwards the JSON description to the service.

```
PROCEDURE SaveEvent
  PARAMETERS
   IMPORT piSectionIndex: INTEGER
   IMPORT piRangeIndex: INTEGER
   IMPORT piEventIndex: INTEGER
   IMPORT piLabel: STRING
   IMPORT piRangeLabel: STRING
   IMPORT piSymbol: STRING
   IMPORT piYear: INTEGER
   IMPORT piShowRange: INTEGER
   IMPORT piEndRange: INTEGER
   IMPORT piLink: STRING
   EXPORT peCalendar: THTMLCODE
RULES
   peCalendar := '<script language="JavaScript">
         calendar.push ({
            "SaveEvent":{
               "Key1":'+STR(nohouse)+',
               "Key2":'+STR(nomem)+',
               "SectionIndex":'+STR(piSectionIndex)+',
               "RangeIndex":'+STR(piRangeIndex)+',
               "EventIndex":'+STR(piEventIndex)+',
               "Label":"'+piLabel+'",
               "EventRangeLabel":"'+piRangeLabel+'",
               "Symbol":"'+piSymbol+'",
               "Year":'+STR(piYear)+',
               "ShowRange":"'+STR(piShowRange)+'",
               "EndRange":"'+STR(piEndRange)+'",
               "Link":"'+piLink+'",
            }
         });</script>'
ENDPROCEDURE
```

When a new event has to be triggered from the Blaise questionnaire this procedure is called and the result is loaded in a string Calendar, which is can be called in a question text as a fill.

If, for example, we have two questions:

```
RC024_kidyob (RC024)
      "@# ^FL_RC024_1 In what year  was ^FL_RC024_2 child born?<br>
```

```
      <i>INT: If year is unknown please .</i>@#" ":
        TYear
RC025_kidname (RC025)
      "@#What is the childs first name?<br>
   ^CALENDAR@#": TKidname
```

And the following routing:

```
Txt_FL_RC024(index, NrOfKids, FL_RC024_1, FL_RC024_2) RC024_kidyob

RC025_kidname

IF RC025_kidname = RESPONSE THEN
   RC025_kidname := UPPERCASE(RC025_kidname)
ENDIF
```

We would add a call to the SaveEvent procedure in the rules:

```
SaveEvent(1, piIndex, 1, RC025_kidname+' born', RC025_kidname , '',
RC024_kidyob, 0, 0, 'Sec_RC.New_Children['+str(piIndex)+'].RC024_kidyob',
CALENDAR)
```

This will make sure the correct JSON structure will be included in the next question text that has a ^CALENDAR fill included.

The calendar is an html table constructed out of independent div containers that have a mouse over event and a mouse click event defined. These divs are generated by an external service. For example a div with its attributes would look like this:

```
<div
 id="calendar_item_4_2_1"
class="calendar_item calendar_item_style_cga_4 calendar_range_4_2"
style="height: 24px; margin-top: 0px; z-index: 43; width: 24pxpx;"
onmouseout="calendar_information_fixed_show();
calendar_highlight_hovered(false, '4', '2');
calendar_information_hover_clear();"
onmouseover="calendar_information_fixed_hide();
calendar_highlight_hovered(true, '4', '2'); calendar_information_hover_set(
'4', '2', '1', 1966, '4_2_1',
'CALENDAR_ITEM');" onClick="calendar_information_fixed_set();">
</div>
```

An `onmouseover`-event calls three functions:

- `calendar_information_fixed_hide();` clears whatever detailed information is currently shown
- `calendar_highlight_hovered(true,'4','2');` highlights the hovered event(s) in the calendar
- `calendar_information_hover_set('4', '2', '1', 1966, '4_2_1', 'CALENDAR_ITEM');` displays the detailed information

An `onclick`-event freezes the current set that is selected. This allows us to move the mouse pointer towards any other event without, accidentally, overwriting the chosen displayed details.

```
<a onclick="javascript: FieldFocused(64, 'Sec_RE.RE002_edfinage', '');"
href="#"> Education </a> (1952)
```

The `FieldFocused`() function will be called and moves the active question to `Sec_RE.RE002_edfinage`. To get the `FieldFocused` function to work, the biPagEvt library should be called from the Style Sheet.

```
<script type="text/javascript" src="libraries\biPagEvt.js"></script>
```

We now established a way of using JavaScript to send JSON strings in Blaise IS. The service that generated the calendar-html was developed in PHP. Unfortunately it was not possible to install the service on the same server and port as Blaise IS. Because of the security design of many browsers it is not possible to call scripts installed on external servers. This issue can be solved by calling a script on the same server that forwards the request. We bridged this problem by using some ASP code that aligned with the Blaise ASP code. The JavaScript code called a service written in ASP (getCalendar.asp), which forwarded the request to a service written in PHP on a different server.

getCalendar.asp:

```
<%
Dim objXMLHTTP
Dim contents
Dim x
Dim URL

contents = ""
For x = 1 to Request.Form.Count
    contents = contents & "&" & Request.Form.Key(x) & "=" &
Request.Form.Item(x)

Next

URL = "http://someserver/lhc_ajax.php"
    Set objXMLHTTP = CreateObject("Microsoft.XMLHTTP")
    objXMLHTTP.Open "POST", URL, false
    objXMLHTTP.setRequestHeader "Content-Type","application/x-www-form-
urlencoded"
    objXMLHTTP.Send contents
    Response.Write objXMLHTTP.responseText
    Set objXMLHTTP = Nothing
%>
```

4. Speeding things up

When testing the questionnaire some problems were encountered; due to the length of the questionnaire and the number of loops and tables loading time exploded. Waiting several seconds up to halve a minute for a new page to show up is simply not workable. In the initial testing phase we found it impossible to work with this. Testing online Blaise questionnaires is in itself a difficult task, because you can never be sure how things will display until the questionnaire is uploaded to a server and viewed with several browsers. With the added problem of the slow loading time, this questionnaire was impossible to test. We implemented a trick to speed things up.

The questionnaire consists of several sections. At the end of each section we set a Timestamp field:

```
TimeStampEnd / "END TIMESTAMP SECTION": TIMETYPE
```

In each section this field is called using the rules:

```
TimeStampEnd.keep
IF (TimeStampEnd = EMPTY) THEN
    TimeStampEnd := SYSTIME
ENDIF
```
This TimeStampEnd is used to set a new section on route only when a previous section is finished.

```
IF (Sec_W.TimeStampEnd <> empty) THEN
      Sec_X
ENDIF
IF (Sec_X.TimeStampEnd <> empty) THEN
      Sec_Y
ENDIF
…
```
This reduced the time it took to generate a page to a reasonable level.

5. Towards a testing environment

The experiences from the development of the online Life History Calendar in Blaise IS triggered some new ideas. Since we found out how to navigate more dynamically, we could use this to create other tools that could help shorten our testing time. In our development of larger online questionnaires we found that especially testing is very frustrating. The layout editor will not always show what a browser will generate. The first challenge is how to let the JavaScript know what fieldname is currently visible. In the html source of a page the name or path is never mentioned. Luckily the Blaise Menu Editor can be used to load this dynamics information in the source code. Using the Blaise Menu Editor we introduce a new control for the language(s) the questionnaire is in.

```
Caption: '<div onload="setPageLog('''+$KEYVALUE+''',
'''+$DATE+''','''+$TIME+''','''+$DATAROOT+''', '''+$FIELDNAME+''',
'''+$QUESTIONTEXT+''')">'
```

The type of caption is switched from ‘Literal text’ to ‘Expression’.
Make sure the ‘Visible’ property is set to True.
The type of the control is set to ‘Label’.

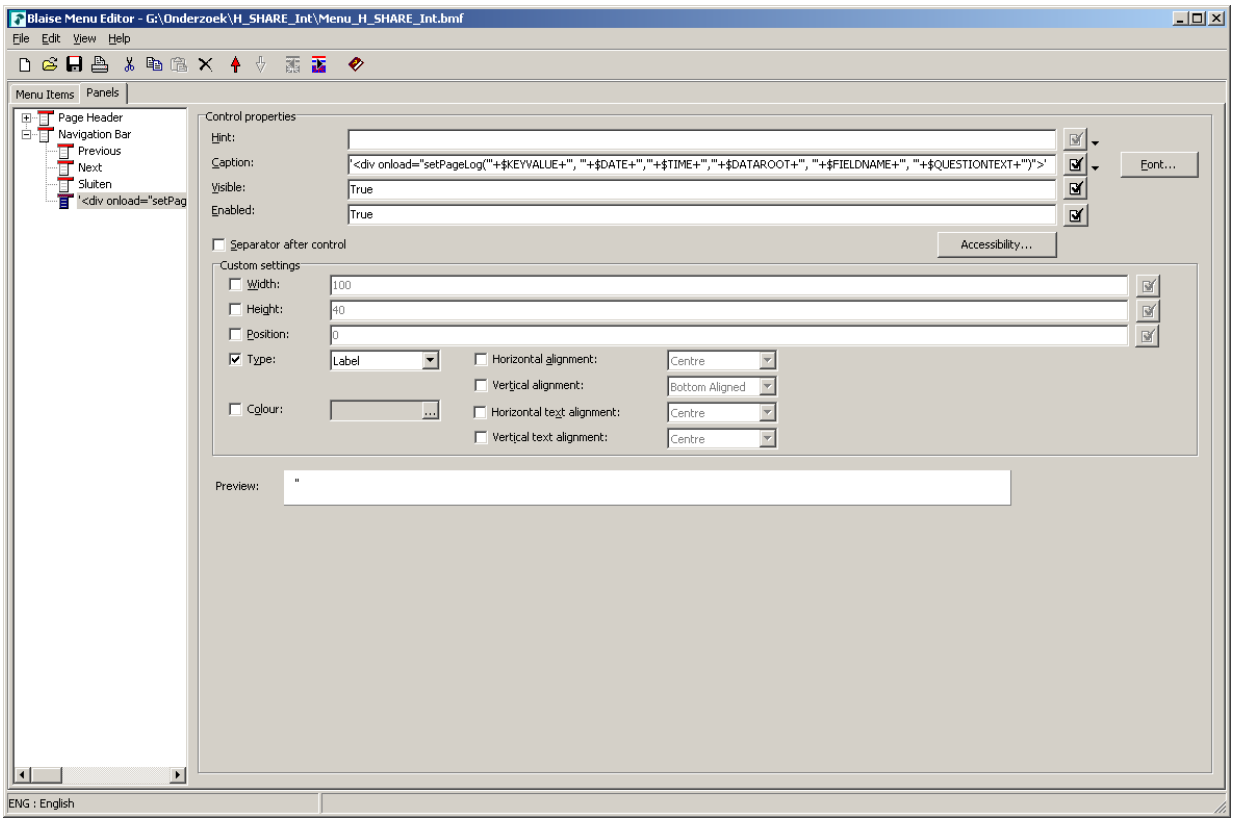

Figure 3: Use the Blaise Menu Editor to feed questionnaire information to the JavaScript

Now each time a new page is loaded in the browser, a JavaScript call will be made to a function setPageLog, sending the key, date, time, dataroot, fieldname, and questiontext.

In our system an Ajax call is made to a service setFieldInfo.php. Its result will be loaded into a div 'testenvironment'

```
function setPageLog(keyvalue, date, time, dataroot, fieldname, questiontext)
{
	var myurl = "setFieldInfo.asp"
	var params =
"dataroot="+dataroot+"&keyvalue="+keyvalue+"&fieldname="+fieldname+"&date="+d
ate+"&time="+time+"&questiontext="+questiontext;

	xmlhttp.open("POST", myurl, true);
	xmlhttp.setRequestHeader("charset", "utf-8");
	xmlhttp.setRequestHeader("Content-type", "application/x-www-form-
urlencoded"); xmlhttp.setRequestHeader("Content-length", params.length);
	xmlhttp.setRequestHeader("Connection", "close");
	xmlhttp.onreadystatechange=function() {
		if (xmlhttp.readyState==4) {
			if (xmlhttp.status==200) {
				setAndExecute('testenvironment',
xmlhttp.responseText);
			}
			document.body.style.cursor = "default";
		}
	}
	xmlhttp.send(params);
}
```

The *testenvironment* div is included in the BlaisePage template in the style sheet of the Interview Page, in our example biHTMLWebpage.xsl

```
<div id="testenvironment " />
```

In biHTMLWebpage.xsl a reference is made to the blaise_tester. js script and to the stylesheet that the testing environment uses:

```
<script type="text/javascript" src="blaise_tester.js" />

<link rel="stylesheet" type="text/css" href="style/tester.css"/>
```

Make sure these new files are available in the BIS

Some ASP files are added to handle the asynchronous JavaScript calls:

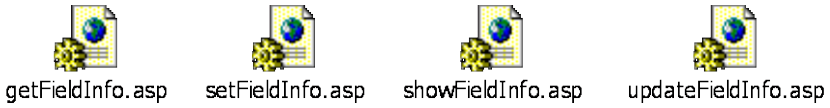

Figure 4: Services as ASP files

Similar to the LHC, we load the testing environment information inline.

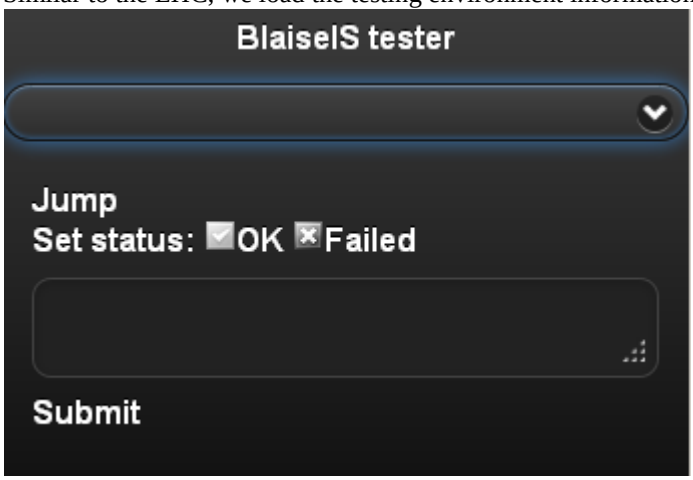

Figure 5: Blaise IS test app

Figure 6: Blaise IS test app with fields found sofar

This app can float and keep track of the route and comments made during the questionnaire test. It seems easily enough to include it into other questionnaires using the method explained in this section. Since it has been recently developed, we did not have the chance to use this in a production environment. The functionality to insert web-apps or other JavaScript applications into online questionnaires strengthens the possibilities Blaise IS has to offer.

6. Conclusion

More and more new ways of complex data measurement are designed and more and more these measurements move online. JavaScript can push the possibilities Blaise IS has, further down this path. Developing questionnaires for Blaise IS is however not very convenient, it sometimes feels like hacking the system to get things working. We hope that Blaise 5 will support the latest techniques better and has easy ways of supporting JavaScript calls and support quick reference testing.