Using the Resource Database to Control Web Security

G J Boris Allan and Peter Stegehuis, Westat, USA

1. Abstract

When mounting most surveys, it is common for web interviews (WIs) to be hosted by a management system (MS). The MS organizes necessary data and then runs a chosen WI. The MS is programmed in some language such as C#, in combination with HTML and similar, and we assume the WI is programmed in Blaise 5. The MS controls web data-security for the WI, and our aim is to program the WI without any need to take into account the MS and associated concerns such as web security of the WI. By adding a standardized single parallel block that does not affect WI program code, and by adding the necessary intelligence to standardized expressions in the WI resource database (the .blrd), the WI can be made secure without modifying the basic Blaise 5 program code.

2. Introduction

There are several fundamental differences between web interviewing on the one hand and CAPI or CATI interviewing on the other that have a significant impact on programming. This is true both for the (Blaise 5) interview itself and for its interaction with a management system.

In CAPI and CATI we have a program on a computer with an interviewer running the program - Blaise 5 as a Windows application – and we can control many aspects because the environment is relatively stable. The interviewer is trained and knows the instrument, special key strokes for DontKnow and Refusal answers, how to use QxQ help and deal with error messages, or how to make remarks when needed. The systems are generally secure, as they are not accessible to outside users.

However, once we deploy a survey to the web, using the Blaise 5 server manager, the extent of our control is greatly reduced:

- Respondents know how they think web pages should behave, and have their own patterns of use. As part of a general trend, mobile phones are becoming used by respondents to answer surveys, and many “web” techniques such as the use of tooltips and other features based on mouse-hover are not valid. We are all susceptible to a cavalier attitude towards any web site we use, and have short attention spans.
- Respondents can be careless (not realizing they are careless) and provides ways for sites can be hijacked by intruders who wish to profit from their intrusions.

Web security is important, but it should not have to intrude into the instrument programming process if possible. That is, the instrument code should be written to do the job of serving the interview design specification and (as much as possible) the code should not be encumbered by trying to recover from accidental or purposeful web misuse.

In the remainder of this paper we will look at a way to separate ‘security code’ from regular Blaise code by using the resource database (.blrd) and a separate management block.

3. Adding a management block

If we want disengage Blaise interview code from having to deal with instrument security, we still need to have some way of coping with events that are out of the normal sequence of affairs. Taking this a bit further, consider a common treatment of interview break-offs, that is, situations where the interviewer hits a key combination (we tend to use CTRL-B key combination), and a new window appears not part of the interview sequence. The interviewer enters a reason for breaking off, exits the window, and the interview ends without existing data being changed. Practices will vary, but a common method is to have a break-off block in the code, distinct from the main interview – a parallel block that may never be used.

Using the break-off block cannot (usually) be left unrecorded within the management of a survey, and so a break-off status is saved to the instrument status (to be counted as part of the management process) – and saving the status is the only change to the main interview code. The instrument status might be a field in a special block for statuses, say, whether CAPI was face-to-face or whether it was by phone, and similar. We call our equivalent a "management" block, partly because some fields in the instrument management block are loaded by values generated in the survey management system – not to be confused with the Blaise survey manager. The survey management system does many things, starting with authenticating users trying to logon to the survey.

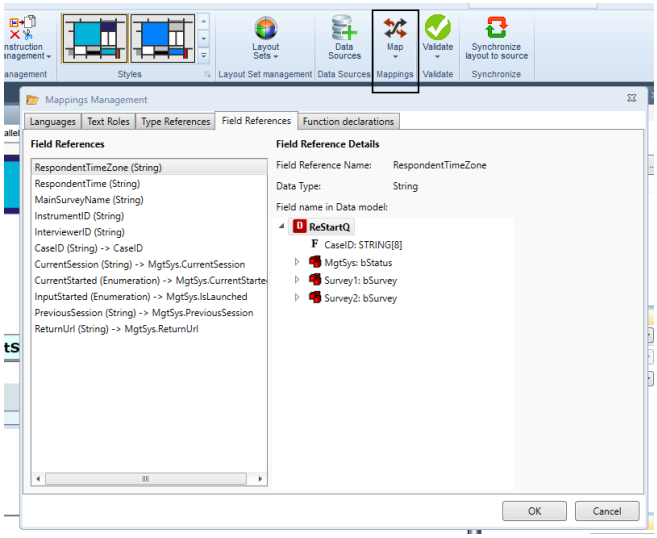
We use the management block (which is never visible to respondent) to store information we need for our security system. For example, here is such a block for one instrument (slightly reformatted):

```
BLOCK BMgtSys
  FIELDS
    // Security fields
    PreviousSession "Previous session GUID"
      : STRING
    CurrentSession "Current session GUID"
      : STRING
    CannotStart "Cannot start interview"
      : STRING[1], EMPTY
    CurrentStarted "Start this interview?"
      : (Yes,No)
    IsLaunched "Set by management system to control access to the instrument"
      : (Yes, No)
    ReturnURL "Management system will write the return URL upon launching the instrument. The instrument
        will use this URL when returning control to the management site. The instrument should not hard
        code the return URL."
      : STRING[100]
    // End of security fields
//
```

```
    // Instrument status fields
    IsReturned "Set to Yes by the instrument prior to return to the instrument (as late as possible).
        Management system consumes the Yes (flips it to No) and then performs extraction."
      : (Yes, No)
    StatusCode "Instrument Status code.  Values are instrument specific and are provided by the project."
      : STRING[2]
    EntryMode "If used, variable can be used by the instrument to behave a certain way, depending on the
        value.  D=Development;  T=Test;  P=Production."
      : STRING[1]
    LastSectionStarted "If used, stores value of the last section that has at least the first question
        answered."
      : STRING[10]
    LastSectionStartedDate "If used, stores date when LastSectionStarted being set"
      : DATETYPE
    LastSectionStartedTime "If used, stores time when LastSectionStarted being set"
      : TIMETYPE
    IsRestart "No immediate need for the variable, but would like to include in management block just in
        case."
      : (Yes, No)
    // End of status fields
ENDBLOCK//BMgtSys
```

In general, status fields are assigned in the Blaise code[1] and stored in the case data, whereas values for fields associated with security (such as PreviousSession), though not assigned in the Blaise code, are still stored in the case data because security-field values are assigned by program expressions in the resource database. The intelligence behind web security is contained in the resource database (a blrd file) because the resource database is capable of far more than designing appropriate layout, or defining what happens when you select a certain button.

Controlling what happens is a result of programming expressions. The connection between expression programming objects (known as field references in the resource database) and corresponding Blaise fields in the current interview session (if necessary) is provided through a mechanism known as mapping. An easy way to look at what has been mapped is by viewing the mapping connector in the Blaise control centre for a simple test instrument (ReStartQ) – the Map/Mappings tab is highlighted by the rectangle:



[1]
```
IF SCR.SCR111b <> EMPTY THEN
    IF ACTIVELANGUAGE = Eng THEN MgtSys.StatusCode := '35' ELSE MgtSys.StatusCode := '36' ENDIF
  ELSEIF (SCR.Box5Route = 'SCR095') AND ((SCR.SCR120 = '1') OR (SCR.SCR120 = SK)) THEN
...
```

As you can see the name of Blaise field in the management block (`MgtSys.IsLaunched`) and the name of the equivalent field reference (`InputStarted`) in the resource database need not be the same (`InputStarted -> MgtSys.IsLaunched`). This means we can change the Blaise field names, remap. If a hacker found that field X was related to resource Y they might try to change values for field X, but the next study could have a new mapping – field W (not field Y) could now be related to resource Y and so previously hacked knowledge would be of no use.

3.1 The blax.layout file

In general, what can be more useful to get a feel for what is happening, is for us to look at the content of the *<Instrument>*`.blax.layout` file, an XML file that makes explicit what might be hidden in the work that goes into establishing layouts for an instrument. If we look at the content of the `RestartQ.blax.`layout file, and – in particular – how the `LayoutSpecFieldReference` items are listed to match the visual presentation of the mappings, and – in general – how layouts are listed in an understandable pattern. Here is a snippet:

```
  <LayoutSpecFieldReferences xmlns="layoutspec">
    <LayoutSpecFieldReference Name="RespondentTimeZone" />
    <LayoutSpecFieldReference Name="RespondentTime" />
    <LayoutSpecFieldReference Name="MainSurveyName" />
    <LayoutSpecFieldReference Name="InstrumentID" />
    <LayoutSpecFieldReference Name="InterviewerID" />
    <LayoutSpecFieldReference Name="CaseID" MappedDatamodelField="CaseID" />
    <LayoutSpecFieldReference Name="CurrentSession" MappedDatamodelField="MgtSys.CurrentSession" />
    <LayoutSpecFieldReference Name="CurrentStarted" MappedDatamodelField="MgtSys.CurrentStarted" />
    <LayoutSpecFieldReference Name="InputStarted" MappedDatamodelField="MgtSys.IsLaunched" />
    <LayoutSpecFieldReference Name="PreviousSession" MappedDatamodelField="MgtSys.PreviousSession" />
    <LayoutSpecFieldReference Name="ReturnUrl" MappedDatamodelField="MgtSys.ReturnUrl" />
  </LayoutSpecFieldReferences>
</LayoutSpecification>
```

Note that the defined field references are the same in the mappings display as they are given as the items labelled as LayoutSpecFieldReference.

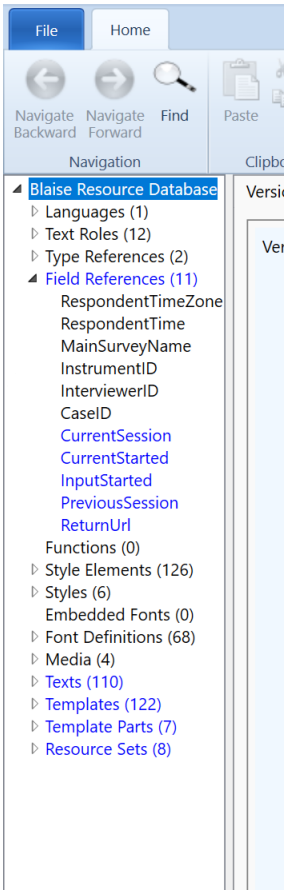
A full listing of the blax.layout is in the Appendix.

3.2 Resource databases and field references

We have mapped some instrument fields to items in the resource database, but what are those items? Those items are field references in the resource database. As we have seen, there are variables (field references) in the resource database that take their initial value from a mapped field in the instrument, and – if the value of the field reference changes – then that change is reflected in the value of the mapped field in the instrument.

When we open the resource database, the field references are listed. Some field references are standard, and appear by default – quite often we have no use for them, and we do not map them to instrument fields. We can add our own custom field references, and for our security system the references are mapped to instrument fields in our management block.

Here are the standard field references (RespondentTimeZone – CaseID) and the added security references (CurrentSession – ReturnUrl) contained in a model security resource database:



The principal places these field references are used are the OnLoad events for different master page templates (less frequently, OnLoaded).

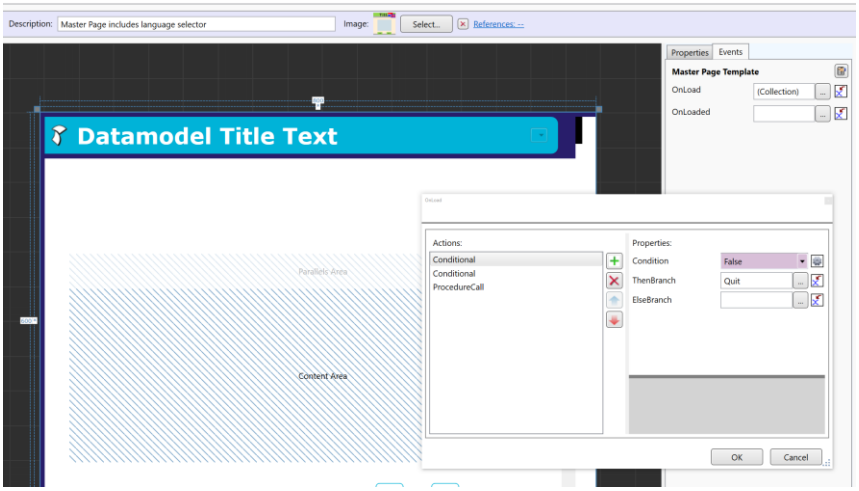
4. Loading web pages

A key idea in web-site design is the idea of distinct web pages. Apart from the case where each successive page overwrites the previous page, web pages can be displayed in new windows (different instances of a browser) or in new tabs (different pages within the same browser instance). One security concern is that we do not pass information in a URL that could be used to create a second instance of an interview in a new tab, or different browser instance.

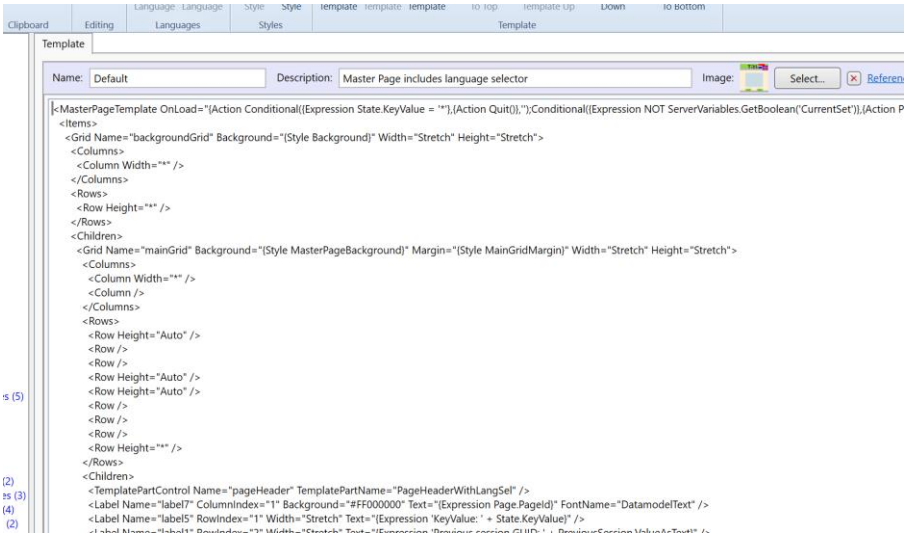
In Blaise 5 there are certain fixed patterns in which information is presented as a page. These fixed patterns are known as master-page templates, that is, designs of how different classes of web page should look and behave. Each different type of master-page template will have different looks and/or behaviors. What we do is attach compound expression code to the `OnLoad` event of each relevant master-page template. In fact we use the same `OnLoad` code for each master page, because we can never be sure where an interview might end. We have to assume that an interview can start/restart at any page.

However, once we have made our security checks, we do want to have a check at every page, so we have to have a mechanism that checks once only per session and doesn't waste time checking again until the interview is restarted.

Here is a master page layout with an expanded `OnLoad` event:



If we change the view of this master page to show the code, we see:



It is difficult to read the `OnLoad` expression above, and so I have reformatted the code to show the structure of the expression in a clearer way.

5. The `OnLoad` expression code

If you look at the master page layout with an expanded `OnLoad` event (above), there are two conditionals followed by a procedure call. This is the first conditional:

```
<MasterPageTemplate OnLoad="
  {Action
    Conditional({Expression State.KeyValue = '*'},
      {
        Action Quit()
      },
      ''
    );
```

That is, if the interview key (case ID) is `*` then quit. This conditional is relevant for when an interview is timed out. We create a case with primary key '*' for the sole purpose of sending control back to the management system in case of a session timeout. The `*` case is created before interviews start as an empty record with only one field (`MgtSys.ReturnURL`) having a value – which sends control back to the management system.

In addition to field references, the expression language knows about server variables, where a better name might be "session" variables. We must provide the name and type of any server variable we use, and when a server session starts the initial value of a server variable is a default. We define a Boolean server variable (`CurrentSet`) and, in the second conditional, if the Boolean value of `CurrentSet` is false (the initial value of `CurrentSet`), we proceed with making assignments to field references, otherwise we skip and do nothing more for the `OnLoad` event.

Later we assign the Boolean true to `CurrentSet`, and so all the expressions under this conditional are skipped.

```
    Conditional({Expression NOT ServerVariables.GetBoolean('CurrentSet')},
      {
        Action ProcedureCall({Expression ServerVariables.SetString('StopReason', 'TimeOut')});
        AssignField('CurrentStarted', {Expression InputStarted.ValueAsText}, '');
        AssignField('InputStarted', '2', '');
        Save();
```

We initialize the server variable `StopReason` to "Timeout" as a default. We don't use `StopReason` for anything at the moment, it is just available in case we make enhancements.

If you look at the mappings for the field reference `InputStarted` then it refers to the field `MgtSys.IsLaunched` (set directly by the management system by means of the API). If `IsLaunched` is true (1) then the instrument has a valid start, and any other value is an invalid start. For security reasons we might want at some point to change the name from `IsLaunched`, and we need only change the mapping.

The value of `InputStarted` is copied to `CurrentStarted`, and `InputStarted` is set to 2. This means that, if the case is restarted someway outside the management system, then the value 2 is copied to `CurrentStarted`. `CurrentStarted` is mapped to `MgtSys.CurrentStarted`. If `CurrentStarted` is 2, then we quit because `MgtSys.IsLaunched` was not true:

```
        Conditional({Expression CurrentStarted.ValueAsText = '2'},
          {
            Action ProcedureCall({Expression ServerVariables.SetString('StopReason', 'NotLaunched')});
            Quit()
          },
          ''
        );
```

We have two field references that refer to session IDs (GUIDs), and we assign what was the previous current session to a field reference `PreviousSession`. We then look at the current session GUID, and assign that to the field reference `CurrentSession`. If a session was broken off without any Blaise knowledge (such as an X-out) then the session could be still alive, and so if the previous session has the same GUID as the current session, something is wrong, and we quit.

```
        AssignField('PreviousSession', {Expression CurrentSession.ValueAsText}, '');
        AssignField('CurrentSession', {Expression State.RuleSessionId}, '');
        Conditional({Expression CurrentSession.ValueAsText = PreviousSession.ValueAsText},
          {
            Action ProcedureCall({Expression ServerVariables.SetString('StopReason',
        'SessionNotClosed')});
            Quit()
          },
          ''
        );
```

Finally, we set the `CurrentSet` server variable to true, so that we don't go through the whole set of assignments again, we have a (gratuitous) check on whether `CurrentStarted` is yes, and we have a final assignment of the exit URL to a server variable (URL).

```
        ProcedureCall({Expression ServerVariables.SetBoolean('CurrentSet',
            NOT ServerVariables.GetBoolean('CurrentSet'))});
        Conditional({Expression CurrentStarted.ValueAsText = 'No'},
          {Action Quit()},
          ''
        )
      },
      ''
    );
    ProcedureCall({Expression ServerVariables.SetString('URL', ExitURL.ValueAsText)})
  }
">
```

You will have noticed that many of our checks end with something like `{Action Quit()}`, and that implies a knowledge of how an interview ends and relates to the main Blaise server. This is part of a larger task, discussed in the paper on session timeouts. [2]

6. Conclusion

Security is a main concern, more so even with web interviewing than it already is with other modes of interviewing. Guarding against unwanted attempts for intrusion in our networks is a prime concern, both for any direct results of hacking as well as damage to reputation, which in turn can lead to a diminished willingness by respondents to engage in future surveys. On top of that, dealing with time-outs and unintended X-out by online respondents, in a way that is both safe and user-friendly is an important issue we have been trying to deal with.

The approach described in this paper is a step in an ongoing process to deal with these new realities.

[2] See "Using the Resource Database to adapt to session timeouts", G J Boris Allan, Joseph Allen, and Siu Wan (IBUC 2020)

7. Appendix: blax.layout

```
<?xml version="1.0" encoding="utf-8"?>
<LayoutSpecification xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Name="ReStartQ" Version="10"
        GenerateAllSections="true" GenerateClientRules="true" UseGenericPages="true">
  <Languages>
    <Language Name="" />
  </Languages>
  <LayoutSets>
    <LayoutSet Name="Interviewing1" ResourceSetName="Large" StyleName="Indigo" RequiredStyle="false"
        DesignHeight="600" DesignWidth="800" ResizeMode="Auto" RouteItemsPerPage="4" RowsPerTable="999"
        ReceiptPageName="Default" IntroPageName="Default" AbortPageName="Default">
      <InstanceLayoutInstructions>
        <RouteItemLayoutInstructions RouteItemName="Survey1">
          <Instructions>
            <NewPageInstruction Locator="Before" />
            <GridInstruction Locator="Before" RouteItemsPerPage="1" />
          </Instructions>
        </RouteItemLayoutInstructions>
      </InstanceLayoutInstructions>
      <Parallels>
        <Parallel Name="">
          <IsGeneric>false</IsGeneric>
        </Parallel>
        <Parallel Name="PRIMARY">
          <IsGeneric>false</IsGeneric>
        </Parallel>
        <Parallel Name="MgtSys">
          <IsGeneric>false</IsGeneric>
        </Parallel>
      </Parallels>
    </LayoutSet>
  </LayoutSets>
  <LayoutSetGroups>
    <LayoutSetGroup Name="Interviewing" DataEntrySettingsName="StrictInterviewing">
      <LayoutSetNames>
        <string>Interviewing1</string>
      </LayoutSetNames>
    </LayoutSetGroup>
  </LayoutSetGroups>
  <RoleReferences>
    <RoleReference Name="Help" />
    <RoleReference Name="Watermark" />
    <RoleReference Name="ToolTip" />
    <RoleReference Name="SpecialAnswer" />
    <RoleReference Name="CategoryGroup" />
    <RoleReference Name="EditMask" />
  </RoleReferences>
  <TypeReferences>
    <TypeReference Name="THeader" />
    <TypeReference Name="TCurrency" />
  </TypeReferences>
  <LayoutSpecFieldReferences xmlns="layoutspec">
    <LayoutSpecFieldReference Name="RespondentTimeZone" />
    <LayoutSpecFieldReference Name="RespondentTime" />
    <LayoutSpecFieldReference Name="MainSurveyName" />
    <LayoutSpecFieldReference Name="InstrumentID" />
    <LayoutSpecFieldReference Name="InterviewerID" />
    <LayoutSpecFieldReference Name="CaseID" MappedDatamodelField="CaseID" />
    <LayoutSpecFieldReference Name="CurrentSession" MappedDatamodelField="MgtSys.CurrentSession" />
    <LayoutSpecFieldReference Name="CurrentStarted" MappedDatamodelField="MgtSys.CurrentStarted" />
    <LayoutSpecFieldReference Name="InputStarted" MappedDatamodelField="MgtSys.IsLaunched" />
    <LayoutSpecFieldReference Name="PreviousSession" MappedDatamodelField="MgtSys.PreviousSession" />
    <LayoutSpecFieldReference Name="ReturnUrl" MappedDatamodelField="MgtSys.ReturnUrl" />
  </LayoutSpecFieldReferences>
</LayoutSpecification>
```