Resolving Text Substitutions

Jason Ostergren, Helena Stolyarova, Danilo Gutierrez
October 2010

13th International Blaise Users Conference
Baltimore, Maryland
The Health and Retirement Study

- The Health and Retirement Study (HRS) at the University of Michigan is a longitudinal study that uses Blaise CATI for its 1 to 3 hour-long biennial interviews of approximately 20,000 participants.

- The HRS interviews are conducted in English or Spanish – both languages reside within the same Blaise instrument.
Text Substitution in the HRS

- The HRS instrument makes unusually heavy use of text substitutions (fills):
  - Most questions make some use of substitutions
  - Text substitutions are often nested inside other text substitutions: fill1 := 'or doing a proxy for ' + fill2
  - Some questions have almost no text that is not part of a text substitution

- The customized question text that results is meant to be more accurate and also less demanding on the interviewer and on the time of the respondent than making substitutions on the fly
Text Substitution in the HRS

- Text substitutions in separate procedures instead of main rules
- Parameters pass results
- Substitutions can pass through more than one procedure
- Why procedures? Aid workflow such as translation, maintenance

```
<table>
<thead>
<tr>
<th>datamodel</th>
<th>PSEUDO-CODE FOR HRS INSTRUMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>fields substitutionarray : array[1..100] of string</td>
<td></td>
</tr>
<tr>
<td>rules</td>
<td></td>
</tr>
<tr>
<td>preparesubstitutions(substitutionarray)   {substitutions exported here}</td>
<td></td>
</tr>
</tbody>
</table>

procedure for preparing common substitutions
if gender = male then
  commonarray[4] := 'he'
exports
  array of common substitutions e.g. [4] = 'he'

contentblock(substitutionarray)

block for some content
imports
  array of common substitutions e.g. [4] = 'he'

fields HeSheSubs : string

procedure unpacking common substitutions
exports dozens of separate strings
poHeShe := piArray[4]
```
Resolving Text Substitutions

• Substitutions can be documented manually by looking at Blaise code and discerning the logic used, but this is problematic:
  ▪ Many parameters
  ▪ Thousands of fills
  ▪ Extremely time-consuming
  ▪ Requires understanding of Blaise code

• Automation is desirable, but hard to achieve
Resolving Text Substitutions

• The next slides will outline the steps necessary to resolve text substitutions automatically in Blaise

• These steps have been used in a few previous attempts at automation by HRS

• Then a discussion will follow of some API features that HRS is using to finally make this automated approach work
General Process I

• Walk through the statements of the Blaise instrument in order and find carets in the question text

> "You should be interviewing Jack, or doing a proxy for him."

Fields

A006_
"You should be interviewing ^Respondents[1].X058AFName, ^txtproxy" : TContinue

• Correctly parse the text for the name of the next substitution variable
General Process II

• The correct instance of the substitution variable must be identified because identically named variables can exist in different blocks

• The program must begin searching for the variable in the immediate block

• If the variable is not found, the program must continue searching upwards through the parent blocks
General Process III

• If the variable was a parameter:
  ▪ identify the name of the variable passed in by the calling block
  ▪ begin the process again based on that variable

• If a fully qualified variable name is provided at any point that ends the search
General Process IV

• Once the substitution variable is identified
  ▪ find and evaluate any values assigned to that variable

• The process of finding these assignments involves walking through all statements looking for:
  ▪ direct assignments to that variable
  ▪ assignments coming from export parameters of blocks or procedures
  ▪ fully qualified references to the variable or generated parameters
General Process V

• When parameter assignments occur:
  - Identify assignments to the export parameter in question inside the other block or procedure
  - This may be a recursive process
Resolving Text Substitutions – API

• The use of parameters is the largest obstacle to resolving text substitutions

• It is vitally important to glean information about parameters from the API in order to trace them
Blaise API Specifics I

Three pieces of information about parameters are vital

• the name of the parameter inside a block or procedure (that is, its name in the parameters section)

• the corresponding name of the variable outside which is passed in as an argument to the block or procedure call

• the direction of the parameter (import, export or transit) – that is, whether the data passed by the parameter is flowing into or out of the block or procedure in question
Blaise API Specifics II

• To obtain this information, it is necessary to look at the StatementText and the Field object for the statement calling the block or procedure.

• If there are parameters involved, the StatementText string will contain the text “{Parameterlist:}” followed by a comma delimited list of names of the parameters on the outside of the block or procedure call.
Blaise API Specifics III

- By splitting this list by commas, an array can be obtained which will correspond to the order of parameters listed in the Fields collection inside the block or procedure.
- These Fields can be obtained from the method `get_DefinedFields()` on the Field object used earlier.
Then it is necessary to loop through this set of fields and obtain each LocalName, which will provide, in order, the inside names that correspond to the variable names from the “Parameterlist”

While looping through the set of Fields, it is vital to collect their ParameterKind, which will require casting them to IBlaiseField2

It may also be helpful to verify that they are parameters by checking that FieldKind == BlFieldKind.blfkParameter
Blaise API Specifics V – BUG!

In rare cases the API functions in such a way as to prevent tracing these substitutions

Procedure called from inside a loop at any structural depth lower than the loop block is treated differently by the API than a procedure called outside of a loop or in the looped block itself

The Field object for the procedure call statement is null in the former case but not in the latter

This prevents us from gathering information about the procedure's parameters as described above

The ideal solution would be for procedures called inside of loops to be handled just like those outside of a loop
Performance

• Parallel XML file when walking through the rules navigator is very helpful

• Much easier to move around an XML document as needed than to try to move around within the Blaise RulesNavigator

• The program moves forward only through the RulesNavigator and then backtracks in the XML document as needed
Blaise Text Example

```blaise
procedure TXTProxySex
   parameters
      import aSex: TGender
      export aString: String
   locals str1, str2, str3: String
   rules
      aString := ''
      str1 := 'or doing a proxy for him.'
      str2 := 'or doing a proxy for her.'
      str3 := 'or doing a proxy for (him/her).'
   if aSex = MALE then
      aString := str1
   elseif aSex = FEMALE then
      aString := str2
   else
      aString := str3
   endif
endprocedure {TXTProxySex}

fields
   P001 "You should be interviewing @r-piRespondents1X05&AFName&r, &r^txtproxy&r": TYesNo
   P002 "Does @r^flhishecr@r have a pension account at &r^flhishecr@r job?"
      "Spanish Does "flheshe have a pension account at "flhishecr job?": TYesNo

rules
   TXTProxySex(Respondents.X05&ASex, txtproxy)
   P001
      procUnpackFillArray(pfillArray, flhishec, flhishecr)
   P002
```
XML File Overview

XML Example

```
<QuestionText>You should be interviewing @r^piRespondents1X058AFName@r, @r^txtproxy@r</QuestionText>

  - <Fills>
    <Fill variable="piRespondents1X058AFName" origin="Respondents.X058AFName"
      fullyqualifiedname="SecPension.piRespondents1X058AFName">FIRST NAME OF INDIVIDUAL</Fill>
    <Fill variable="txtproxy" origin="SecPension.txtproxy"
      fullyqualifiedname="SecPension.txtproxy">/[or doing a proxy for him.]/[or doing a proxy for her.]/[or doing a proxy for (him/her).]</Fill>

  </Fills>
```
Simple Questionnaire

- The Simple Questionnaire (SQ) is a formatted output text file designed to document the set of questions in the order and the form they are presented to a respondent, including question wording with accurate text substitutions.
Simple Questionnaire 2

By construction, each question is represented by a separate node in the XML file, with the following attributes attached to the node:

- Field Name
- Long Field Name
- Descriptors
- Question Text with resolved text substitutions
- Codeframes with resolved text substitutions
- Data type
- DK/RF flag
- Empty Flag
• BLAISE INSTRUMENT

Fields

A006_
"You should be interviewing @r^piRespondents1X058AFName@r, @r^txtproxy@r": TContinue

• SIMPLE QUESTIONNAIRE

SecA.StartInterview.A006_

You should be interviewing [FIRST NAME OF INDIVIDUAL] [R LAST NAME],
/[or doing a proxy for him.]/[or doing a proxy for her.]/[or doing a proxy for (him/her).]

1. Correct R - go on
Potential Uses of XML

- Simple Questionnaire, text file with:
  - Field Name
  - Long Field Name
  - Descriptor
  - Question Text in all Languages with resolved text substitutions
  - Codeframes in all Languages with resolved text substitutions
  - Data type
  - DK/RF flag
  - Empty Flag

- Relational Database

- HTML file

- Support Table of Fills for Michigan Questionnaire Documentation System

- Automated Text Comparisons

- Field Effect Finder
Other Benefits

- Producing metadata for quality checks and end-user documentation by allowing us to:
  - produce field descriptors and quickly check for duplicates and errors in our large instrument
  - produce field names in ask order sequence for checking integrity and completeness in documentation
  - produce question text and codeframe text and check them for omissions and errors
Conclusion

• New product to save time and increase accuracy in metadata processing of text substitutions for IRB documentation to obtain study approval

• Provide a source of metadata with text substitutions in the XML which could be exploited by small, easy-to develop programs for a variety of uses
Contact Information

- Jason Ostergren
  - jostergr@isr.umich.edu
- Helena Stolyarova
  - hstolyar@isr.umich.edu
- Danilo Gutierrez
  - danilog@isr.umich.edu

Questions

Thank You