Some Ideas On Securing And Processing A Blaise IS Survey

Rob Groeneveld, Statistics Netherlands

1. Introduction

In Blaise IS, two modes of survey are possible: interview mode and form mode. In interview mode the respondent interacts with the survey in much the same way as a respondent who answers a DEP interview. In form mode, the respondent answers all questions and sends back the form to the Blaise IS server. No interaction occurs in form mode, for instance, no routing is possible and checks are limited to range checks. When using interview mode in a lengthy survey, the question arises: can a respondent come back to a partially completed form? This is possible if one accepts a slight security risk. This paper presents a few suggestions on how to keep the interview data confidential and enable resumability. Moreover, a diagram of a possible processing sequence for this kind of survey is presented.

2. Ensuring confidentiality and resumability

We distinguish security and authentication. *Security* means that the data sent over the Internet are inaccessible to others. This can be accomplished using the Secure Sockets Layer (SSL) protocol. *Authentication* means that only the person intended has access to certain specified data. This paper deals with authentication according to a certain method which can be characterised as ‘Questionnaire based’. A number of other methods are possible for authentication, including methods using the facilities of the operating system. The advantage of the method presented in this paper is that it only requires an understanding of Blaise and Blaise IS. The disadvantage of this method is that it changes the datamodel — the datamodel is no longer identical to the model used in face-to-face and other types of interviewing.
The Secure Sockets Layer protocol encrypts data transported via the Internet, keeping them confidential. The SSL is an option in Blaise IS which should always be used. If a respondent is able to return to a previously partially filled form, stored in a record on the Blaise IS server, the issue of authentication is of even more concern. If one respondent can return to his form, other respondents can also retrieve that form, unless it is made difficult for them to do so.
We present here a technique whereby a key field is added to each record with individual ID numbers generated randomly, so as to have a key field which is difficult to guess. We suggest using at least 10,000 times the number of respondents as a reservoir from which to draw the ID numbers. The chance to match an ID number by guessing is thus comparable to the chance of typing the correct four-number PIN in an ATM with somebody else’s bank card.
The Blaise setup used for the survey is to be extended with ID Number and Password fields. The ID Number becomes the primary key. The password provides additional security.The type TPassword is introduced in order to mask the input with a password character.
At the beginning of the datamodel we add these sections:

```
PRIMARY
  IDNUMBER
TYPE
  TIDNUMBER = STRING[12]
  TPassword = STRING[4]
FIELDS
  IDNUMBER   "Enter the 12-digit number that was sent to you":
    TIDNUMBER
  Password: TPassword
```

The type and field sections can be interspersed with types, blocks and fields of the original datamodel.

3. Generating ID numbers and passwords

We use Manipula to populate the added fields. A Manipula statement to generate an arbitrary string with length 12 containing only digits is the following:

```
  FOR I := 1 TO 12 DO
    IDNUMBER := IDNUMBER + STR(RANDOM(10))
  ENDDO
```

In which I is an integer auxfield and IDNUMBER is the field with the ID.
As a further enhancement to confidentiality, the password is generated randomly:

```
  FOR i := 1 TO 4 DO
    Password := Password + CHAR(97 + RANDOM(26))
  ENDDO
```

This command generates a password consisting of four lowercase letters. Both ID number and password are sent to a respondent by email, by normal mail or automatically using the Mail API or the Outlook library. When a respondent starts the survey, he can change his password immediately, so even people who happen to read the letter with the ID number and the initial password cannot reopen his record. The password change can be accomplished with normal Blaise AUXFIELDS and RULES:

```
AUXFIELDS
  EnterPW  "Enter your password":  TPassword
  ChangePW "Do you want to change your password?": (no, yes)
  NewPW    "Enter your new password": TPassword
RULES
  EnterPW
  Password.KEEP
  EnterPW = Password "You entered an incorrect password"
  ChangePW
  IF ChangePW = yes THEN
    NewPW
    IF NewPW = RESPONSE THEN
      Password := NewPW
      EnterPW := NewPW
    ENDIF
  ENDIF
  NEWPAGE { This protects the rest of the questionnaire }
```

The field Password is a datafield stored in the record. It is never asked. The field EnterPW is used to ask the respondent for his password. The check EnterPW = Password tests if the typed password is equal to the stored password. If not, an error message is presented and the password has to be typed again. The next field, ChangePW, and the next page(s) of the interview cannot be entered until the password is correct. If the password is correct, the next question is whether the password must be changed. It is best to change the password the first time the interview is run. If the respondent wants to change his password, the field NewPW is asked. When this field is filled in, the fields Password and EnterPW are updated so the check remains satisfied. The new password is stored in the record and must be entered correctly the next time the interview is opened. Only if the check is satisfied is it possible to proceed to the next page due to the NEWPAGE command. The survey questions proper start from this page. All password fields, normal and auxiliary, are of type TPassword, which is a string of 4 characters (in this case). This type is defined as a password type in the Datamodel Properties, the password character being for instance '*'. A PRINT file called Report.txt is also defined in the Manipula setup which reports the ID Numbers and passwords in a convenient format. The only field in this print file is a simple line of characters:

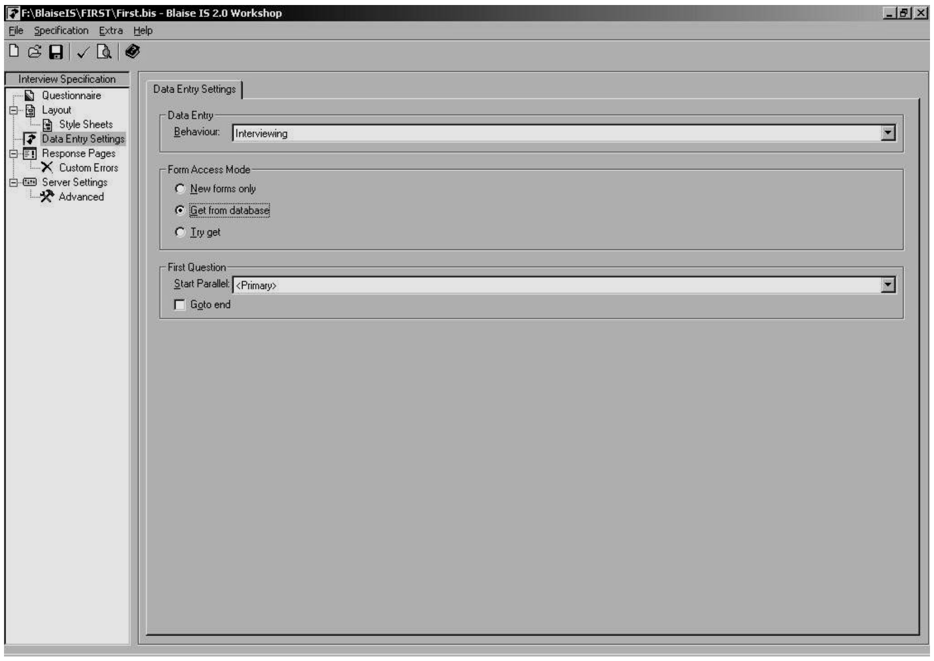
```
Line1: STRING[255]
```

The Manipula setup writes the Name, ID Number and Password to this field:

```
  Line1 := FORMAT(Name, 20) + IDNUMBER + '  ' + Password
```

Furthermore, some layout features are added to the report file. This file is of course only intended for the person managing the Blaise IS Survey.

Blaise IS must be configured in such a way as to retrieve only records with an existing primary key. This is done in the Data Entry Settings in the Blaise IS Workshop:



Data Entry
Behaviour: Interviewing

Form Access Mode
( ) New forms only
(•) Get from database
( ) Try get

First Question
Start Parallel: <Primary>
☐ Goto end

Figure 1. Data Entry Settings in the Blaise IS Workshop

With the option Get from Database the respondent is only allowed to access a form when his key is in the database. If his key is not in the database, he gets an error message.
All records generated with Manipula (empty except for the ID number, password and data available beforehand) are stored on the Blaise IS server in the Data subfolder. Only people who know a valid ID Number and password can access a record. If a respondent wants to leave the survey and come back to it later, he can again retrieve his record using the same ID Number and the (possibly new) password. Another setting is useful in this regard: ‘Goto end’ in the above screen, when checked, causes the questionnaire to jump to the first unanswered question.
There are a number of other methods for authenticating a respondent of a Blaise IS survey. More information can be found in the Blaise IS Online Assistant.

4. Processing a survey with ID Numbers

Assuming that some respondent data (for instance, registration number, name and address) are already stored in a table in a SQL Server database in the statistical office, the records can be read from this database with Manipula, extended with an ID Number and password as above and stored in a database which is used in the survey in Blaise IS. This can be a Blaise database, but it can also be another database: Oracle, SQL Server, MS Access or a similar system. The database can even be located on another server than the Blaise IS server. In the processing diagram below we assume the records which will receive the answers to the questions are stored in a Blaise database (.BDB file) on the Blaise IS server. The ‘original’ datamodel is called FIRST, the same model extended with an ID Number and Password is called FIRST_BL_IS.

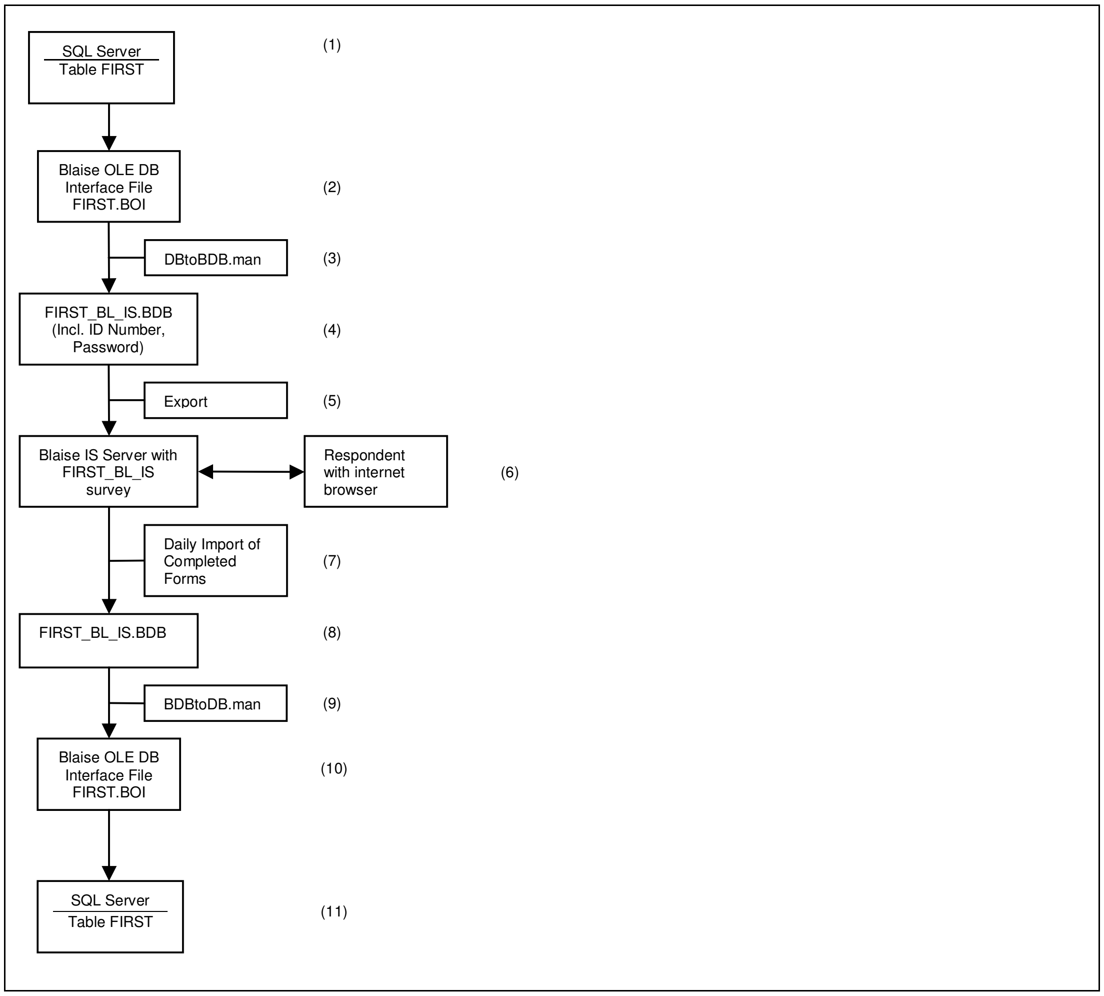

SQL Server
Table FIRST
(1)
Blaise OLE DB Interface File FIRST.BOI
(2)
DBtoBDB.man
(3)
FIRST_BL_IS.BDB (Incl. ID Number, Password)
(4)
Export
(5)
Blaise IS Server with FIRST_BL_IS survey
Respondent with internet browser
(6)
Daily Import of Completed Forms
(7)
FIRST_BL_IS.BDB
(8)
BDBtoDB.man
(9)
Blaise OLE DB Interface File FIRST.BOI
(10)
SQL Server
Table FIRST
(11)

Figure 2. Flowchart: Processing a Blaise IS Survey with an RDBMS

The steps illustrated are:

1. A table on the SQL Server holds data on respondents, for instance, registration number, name and address. Our example table is called FIRST.
2. With the Blaise DataLink a BOI file is constructed which opens this table to processing with Blaise and Manipula.
3. A Manipula setup transports the data in the table to a Blaise database and assigns ID Numbers and passwords at the same time.
4. The result is a Blaise database with ID Numbers and passwords and the name, address and other data from the respondents database.
5. This database is exported (perhaps through a firewall or by email, depending on the ICT configuration of the office) to a database on a Blaise IS Server. This step is not necessary if the file (4) is on the Blaise IS Server.
6. The survey is run on the Blaise IS server. A respondent uses his internet browser to answer the questions in the survey. If necessary, he can come back to his record later.
7. The completed records are moved back to the Blaise database within the firewall and deleted from the Blaise IS server. Incomplete records remain on the server for some limited time, let's say, one week. Complete forms are inaccessible to other respondents once they have been moved. Depending on the configuration, this step can be a Manipula setup or a more elaborate step involving email etc.
8. The records are now on the Blaise database within the firewall.
9. The records are transported back to the SQL Server database, updating the fields in the original table with the data from the respondents, or storing the data in a new table, using

10. the BOI file of step 2.
11. The records with the completed fields are now in a SQL Server table.

5. Summary

We presented some ideas on authenticating users using an addition to the basic Blaise datamodel. Also, we described a possible process for a Blaise IS survey in Interview mode which uses this method. A respondent can access his own record, but not other records. He need not answer all questions in one session, but he can resume the survey using his own partially completed form for some time after first opening it.

Exploring the capabilities of Blaise IS

James O'Reilly, Westat, USA

1. Introduction

While questions remain about the effectiveness of web surveys, interest in web-based surveys has grown among researchers in government, private, and academic organizations. Some clients are pursuing exploratory efforts to learn whether the often discussed challenges--sampling, respondent cooperation, item non-response, self- rather than interviewer-administration, and the vicissitudes of the web environment--can be overcome or at least managed at a level that produces something useful. Others are interested in mixed-mode surveys, usually CATI and web. Still others want to recast existing web surveys: enhancing capabilities or increasing cost effectiveness, or expanding the scope.

Westat has integrated the internet into its survey work for a number of years, including conducting web surveys. Recently, growing demand and changing web research designs stimulated a company effort to examine our web survey processes and technology. One key goal is to evaluate and, where appropriate, implement web surveys using Blaise IS.

Westat's web surveys have been developed mainly using ASP, Microsoft's scripting environment for generating and processing HTML pages. ASP provides a powerful set of capabilities including easy integration with SQL Server and other relational databases. However, along with its power and flexibility, ASP is a procedurally-oriented, relatively low-level development language. This means custom programming and significant maintenance and support challenges.

Westat's ASP development team has built authoring systems to enhance our capabilities--generating much of the ASP code and reusing standard code modules. Still, it is clear that the ASP approach is problematic for some important types of surveys, for example multi-modal and more complex questionnaire designs.

Of course, all these things fit Blaise IS to a tee, in theory. Blaise is intrinsically multi-modal; it handles complexity naturally; its rules engine provide unmatched routing control; and our Blaise experience in CAPI, CATI, and CASI surveys could be applied to web efforts. But is Blaise IS ready to deliver, ready for prime-time with the architecture, scalability, usability, and extensibility required? That's what our effort was designed to address.

Our major activities were:

- User interface testing and experimentation
- Implementing XSL stylesheet extensions
- Exploring instrument development methods
- Developing a training course
- Designing and planning for server-side management and support systems

We will cover the first three of these here. Most of the work was done using IS 2.0 build 161 of 21 November 2003. As we discovered some of the limitation of IS 2.0, we awaited impatiently the release of the Blaise 4.7 beta so we could test some of the large and small enhancements. Since the beta release on July 30th we have been working intensively on version 4.7 and will report on that also.

2. Web interview user interface

In evaluating the user interface (UI) capabilities of IS, the core issue is that the user may often have very limited familiarity with computers, web browsers, and how to interact with them, whereas Windows Blaise UI is designed for and used mainly by trained, experienced interviewers. We focused first on elements of relatively simple web surveys of individuals, using the NSSE2003 example. We found IS 2.0 handled well scalar items in dropdown lists, enumerated, string, numeric and open text. Also it implemented nicely grouped enumerated series of questions.

The technique implementing the enumerated grouping was a critical improvement to IS. It revealed how, using naming conventions for field response types in the standard Blaise datamodel, it is possible to instruct the XSL stylesheet to change the way one or more items are rendered in the HTML page sent to the browser. We discuss this in a later section.

We found problems with the usability of complex scalar items such as date and time fields. In Windows Blaise edit masks help the user understand what components of the field to enter. Masks are not allowed in the HTML text entry control. Again extensions to the XSL stylesheet are the solution.

Other UI problems in IS 2.0 included:

- Conditional items displayed on the next page, after a trip to the server
- Limited functionality for multiple language interviews

Both of these are fixed in Blaise 4.7.

Establishment survey prototyping raised other issues, including

- Limited flexibility for controlling the table display
- Difficulty with form-like entry of information with many fields on a page and more than one field on a line desireable.

2.1 XSL stylesheet extensions

The architecture of Blaise IS provides great benefits in terms of making the Blaise system's superior features--high-level, structured language using blocks and arrays to implement complex instruments; unmatched rules engine; capable proprietary database; and other elements--available for web interviewing. To do this IS controls the process much more strictly than other web system which typically allow the use of custom JavaScript and other modules to tailor and customize the web application.

The result until recently was that using Blaise IS meant accepting a more limited and dull user interface compared to what one could expect from an ASP or other native web development approach. For some web applications, such as CATI or data editing, with trained staff as users, the tradeoff seems easy to accept. But for many customers the most important part of web survey work is the ability to conduct self-administered surveys—having study subjects do the interview themselves.

For such untrained, voluntary users with widely varying web experience a limited and simple user interface is not likely to be desireable. Expectations of many clients on the look and feel of web applications are for polished, flexible pages.

Fortunately, the IS architecture uses XSLT (Extensible Stylesheet Language for Transformations), usually referred to as XSL. XSL provides a way in IS to extend and refine key functions and interface elements in web surveys. This extensibility potential is different from and in many ways superior to that available in traditional Blaise (DLLs or the BCP).

During a Blaise web survey each time the IS server is ready to communicate with the user's browser, IS uses the API to generate an XML (Extensible Markup Language) document with the appropriate data and metadata on the fields, texts, layout, and other facets of the current situation. The XML document's rich content provides such information as the field's type which allows the stylesheet to process in various ways.

The document is processed by the MSXML parser using the interview XSL stylesheet, biHTMLWebPage.xsl, or a substitute. This process transforms the XML information into the HTML page that is sent to the user.

API > XML > MSXML + XSL > HTML

Figure 1: Blaise IS server steps

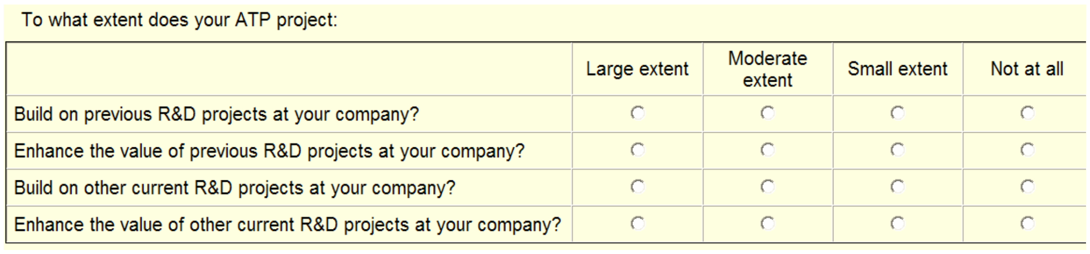
As we prototyped surveys in IS 2.0, we identified a number of user interface elements that we needed but could not render with the system as-is. Our engineers studied the system and concluded that we could generate many of the requested elements by adding functions in the interview XSL stylesheet, following the model developed by the Blaise team for displaying a series of enumerated items as rows in a table with the question text in the stub and the enumerated choices as the columns.

Here's an example. In the datamodel TYPE section a TGroupExtent is defined (Figure 2) and applied to the four fields.

```
TYPE
  TGroupExtent = (large "Large extent", Moderate "Moderate extent", Small "Small extent",
    notatall "Not at all")
FIELDS
    BUILD_PREV "Build on previous R&D projects at your company?": TGroupExtent
    ENHANCE_PREV "Enhance the value of previous R&D projects at your company?": TGroupExtent
    BUILD_CURR "Build on other current R&D projects at your company?":TGroupExtent
    ENHANCE_CURR "Enhance the value of other current R&D projects at your company?":TGroupExtent
```

Figure 2: Grouping code

The interview stylesheet has been programmed to check for the answer types of fields, looking for a series of enumerated fields with the same type name and the prefix TGroup. When a set is found the stylesheet instead of displaying the series as individual items in sequence on the page, renders them in a table (Figure 3).



To what extent does your ATP project:

	Large extent	Moderate extent	Small extent	Not at all
Build on previous R&D projects at your company?	○	○	○	○
Enhance the value of previous R&D projects at your company?	○	○	○	○
Build on other current R&D projects at your company?	○	○	○	○
Enhance the value of other current R&D projects at your company?	○	○	○	○

Figure 3: Grouped enumerated display

This model made clear that a wide range of web-standard user interface elements could be added to Blaise IS using only the standard Blaise datamodel syntax and extensions in the XSL interview stylesheet.

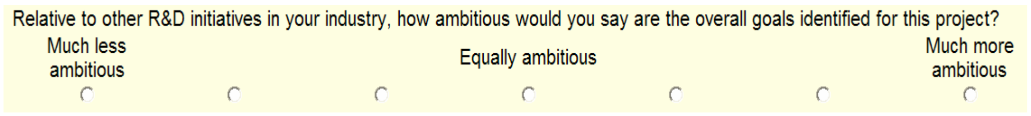
We then began to explore the details of implementing this capability as we prototyped instruments and found web interview interface needs that could not be met as fully as we wanted using only the standard Blaise modelib.

2.2 Scales

Using the modelib one can display Likert-type scales horizontally but not with the labels displayed above the points rather than to one side. We requested this of the Blaise IS developers which they quickly provided. In the datamodel one uses a TYPE with a prefix of TScale (Figure 4) and a corresponding function in the interview stylesheet to produce the display (Figure 5).

```
TYPE
  TScaleAmbitious = (MuchLess "Much less ambitious", Less " ", SomeLess " ",equal "Equally ambitious",
    SomeMore " ", More " ", MuchMore "Much more ambitious")
FIELDS
    PROJ_AMBITIOUS "Relative to other R&D initiatives in your industry, how ambitious would you
      say are the overall goals identified for this project?":TScaleAmbitious
```

Figure 4: Likert scale extension code


Relative to other R&D initiatives in your industry, how ambitious would you say are the overall goals identified for this project?

Much less ambitious			Equally ambitious			Much more ambitious
○	○	○	○	○	○	○

Figure 5: Scale display in IS

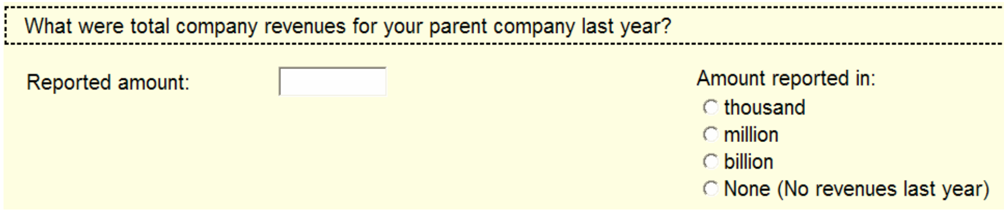
2.3 Quantity-Unit

Quantity-unit questions are important to many surveys. We need to know some quantitative information ("How much are you paid?") and the respondent may think of the concept in terms of so much per hour, per week, per two weeks etc. Using standard Blaise with an interviewer, the two elements are different fields and the interviewer may be able to complete both based on the response received to the first part. Or a follow-on question about the unit may be required.

In self-administration this compound concept can be challenging. So we wanted the visual representation to link the two fields more clearly than a two-step dialog displayed vertically on the page. We specified that two successive fields of types prefixed with TQuanUnit would be displayed side-by-side. Our XSL specialist added this to the stylesheet. The datamodel code and display are below.

```
TYPE
  TQuanUnit_Unit = (thousand, million, billion, noRev "None (No revenues last year)")
  TQuanUnit_Quan = 0..999999
AUXFIELDS
      introrev "What were total company revenues for your parent company last year?":string[1], empty
FIELDS
      RevAmount "Reported amount:"/"Reported amount:": TQuanUnit_Quan
      RevUnit   "Amount reported in:"/"Amount reported in:": TQuanUnit_Unit
RULES
      introrev
      revamount
      revunit
```

Figure 6: Quantity-unit code


What were total company revenues for your parent company last year?

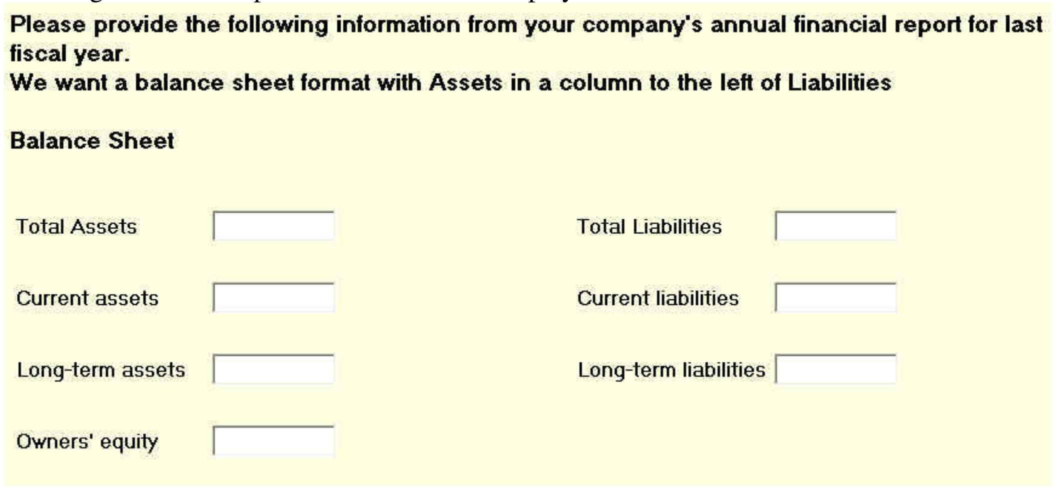
Reported amount:

Amount reported in:
○ thousand
○ million
○ billion
○ None (No revenues last year)

Figure 7: Quantity-unit display

Other user interface extensions added using the stylesheet include:

- Enabling two dual input lines to able to display assets and liabilities in two columns.

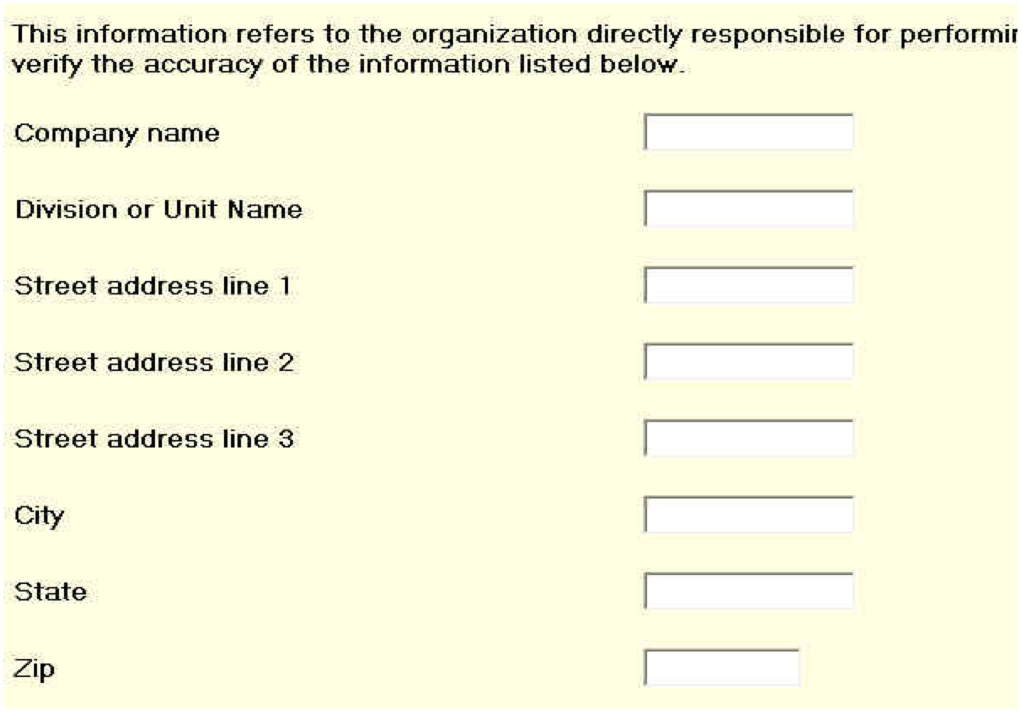


Please provide the following information from your company's annual financial report for last fiscal year.
We want a balance sheet format with Assets in a column to the left of Liabilities

Balance Sheet

Total Assets		Total Liabilities	
Current assets		Current liabilities	
Long-term assets		Long-term liabilities	
Owners' equity			

- Altering line spacing to show more elements on a page.



This information refers to the organization directly responsible for performir
verify the accuracy of the information listed below.

Company name	
Division or Unit Name	
Street address line 1	
Street address line 2	
Street address line 3	
City	
State	
Zip	

- Adjust the column widths of grouped enumerated sets so that the widths the same.
- Improved control of the width of text
- Increasing the size of the radio button and check box controls to improve ease of use
- Changing the menu and parallel tab look.

2.4 Applying custom stylesheet extensions

These extensions were added to the default IS 2.0 web interview stylesheet, biHTMLWebInterview.xsl, following XSL programming standards to prevent our references from conflicting or interacting with other functions in the stylesheet. This is done by defining the Westat namespace (Figure 8) so that all variables related to the extensions are identified by the namespace

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:westat="http//www.westat.com"
version="1.0">
```

Figure 8: Westat namespace defined

To use the Westat extensions, developers start with the extended interview stylesheet. Working in the IS workshop and IS stylesheet editor the standard instrument customization is done—changing header and footer appearance, fonts, images, and the like—and saving a instrument-specific version of the extended stylesheet.

If the developer wants to change the default settings of the Westat extensions, the IS stylesheet editor can't be used. Instead a text editor or similar tool is needed to access the XSL code and enable or disable specific functions (Figure 10) or change parameters for the functions (Figure 10).

```
<!-- Westat start -->
<!-- OpenSource XSLT flow-control variables - all contributors -->
<xsl:variable name="westat:EnableQuantityUnit">true</xsl:variable>
<xsl:variable name="westat:EnableLineSpacing">true</xsl:variable>
<xsl:variable name="westat:EnableInputLineAlign">true</xsl:variable>
<xsl:variable name="westat:EnableEnumColWidth">true</xsl:variable>
<xsl:variable name="westat:EnableDualInputLine">true</xsl:variable>
<xsl:variable name="westat:EnableTabletStyle">true</xsl:variable>
<xsl:variable name="westat:EnableSpecialEffects">no</xsl:variable>
<xsl:variable name="westat:EnableMenu">true</xsl:variable>
```

Figure 9: Enabling/disabling extension functions

```
<!-- OpenSource Westat -->
<xsl:variable name="westat:DualInputLinePrefix">TDualInputLine</xsl:variable>
<xsl:variable name="westat:QUPrefix">TQuanUnit</xsl:variable>
<xsl:variable name="westat:LSCellSpacing">0</xsl:variable>
<xsl:variable name="westat:LSCellPadding">0</xsl:variable>
<xsl:variable name="westat:FieldLabelWidth">40%</xsl:variable>
<xsl:variable name="westat:FieldNameWidth">40%</xsl:variable>
<xsl:variable name="westat:InputLineWidth">60%</xsl:variable>
<xsl:variable name="westat:TabletRadioButtonWidthHeight">width:26px; height:26px</xsl:variable>
<xsl:variable name="westat:TabletCheckBoxWidthHeight">width:26px; height:26px</xsl:variable>
<xsl:variable name="westat:TabletButtonWidth">150px</xsl:variable>
<xsl:variable name="westat:TabletButtonHeight">40px</xsl:variable>
<xsl:variable name="westat:HighlightColor">cyan</xsl:variable>
<xsl:variable name="westat:ParallelTabBackgroundColor">yellowgreen</xsl:variable>
<xsl:variable name="westat:ParallelTabFontWeight">bold</xsl:variable>
<xsl:variable name="westat:ParallelTabFontFamily">arial</xsl:variable>
<xsl:variable name="westat:CurrParallelTabFontWeight">bold</xsl:variable>
<xsl:variable name="westat:CurrParallelTabColor">rgb(120,120,0)</xsl:variable>
<xsl:variable name="westat:CurrParallelTabBackgroundColor">gold</xsl:variable>
<xsl:variable name="westat:MenuButtonBackgroundColor">darkolivegreen</xsl:variable>
<xsl:variable name="westat:MenuButtonColor">yellow</xsl:variable>
<!-- Westat end -->
```

Figure 10: Extension functions parameters

2.5 Technical and organizational challenges of XSL extension

XSL stylesheets in Blaise IS open a significant avenue for extending the already powerful and broad capabilities of the out-of-the-box product. Some key features of this approach include:

- XML and XSL are industry-standard Internet technology being taught and learned widely, and not something restricted to a small niche. Courses, books, and other materials are readily available. So investing, building and applying XSL to web applications in Blaise is feasible at least for a larger organization.
- The XSL language is structured to preclude or minimize the chances of side effects in which one function or extension disrupts standard functions or other extended functions. So, following established development standards, extensions can be added with little risk of corrupting the standard Blaise XSL functions or other third-party extensions.
- Because of the richness of the XML document Blaise generates and passes, the XSL process has access to a large set of elements for its transformations and generation of the page to be sent to the browser. So the range of things that are accessible through the stylesheet is quite broad and substantial.

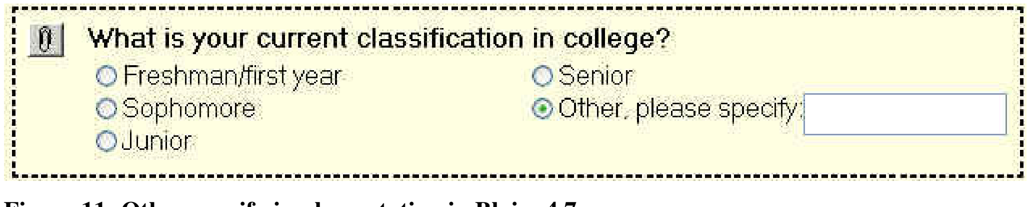
There's always a catch, of course. In this case it's that the power and potential of XSL for tailoring Blaise IS interviewing pages requires someone with serious technical skills in XML/XSL, and that person will have to become knowledgeable about the Blaise object architecture and its representation in XML.

The web interview stylesheet biHTMLWebPage.xsl for IS 2.0 has 5,335 lines and that in the 4.7 beta has 7,163 lines. Obviously, working in such a complex program requires skill and strong understanding of the details of the XSL and Blaise languages.

Does this mean that for all practical purposes extending IS web interviewing is only realistic for large organizations? This seems likely in most situations. At the same time, based on our work so far it seems that only a limited number of web survey interface elements are needed to make the system able to meet a very high percentage of web interview needs.

While the technical details of XSL programming with Blaise is far beyond the understanding of this writer, the Westat experience so far is that adding functions in the stylesheet is not as difficult and time-consuming a process for the skilled, prepared XSL developer as might be feared. Therefore we might be able to expect that the Blaise team will be willing to evolve its web interview stylesheet to add interface capabilities that users and the team agree are of general use.

Blaise 4.7 includes a powerful example of what can be done with XSL. The specify half of the other-specify is immediately activated and displayed when the "Other" choice is selected.


What is your current classification in college?

Freshman/first year
Sophomore
Junior
Senior
[✓ Other, please specify:]

Figure 11: Other-specify implementation in Blaise 4.7

No doubt the Blaise team is as excited as the IS users at the opportunity to continue to expand the usability of IS with such XSL innovations.

2.6 Wish-list of web interview capabilities extension via XSL

Here is a list of user interface extensions of general use that this writer would like to have. They are based on the work of Mick Couper, a leader in research on web interviewing and user interface issues.

Text input areas with labels linked to the text input box.

A currency symbol before or after the field.
$ [______]

A '%' symbol before or after the field.
[______]%

Specification: using a prefix of TSymLabel, fields of this prefixed type will get a text label from the field text string "[L|R|A|B][symbol]", e.g.

/"L$" {Place a "$" symbol to the left of the text entry field}
/"R%" {Place a "%" symbol to the right of the text entry field}
/"B(___) ___-_____" {Place the format string "" below the text entry field}

```
TYPE
TSymLabelPay: 1000..100000
 TSymLabelPct: 10..100

FIELDS
Salary "What do you get paid?"/"L$": TSymLabelPay
 Taxes "What percent of your pay goes to taxes?" / "R%": TSymLabelPct
```

Figure 12: Example code for labeled text entry controls

Dates in three fields with separator and labels

__ / __ / ____
mm dd yyyy

Specification: using the prefix TFmtDate, three fields of this prefixed type will be displayed in the order asked with the field text shown below as a label. The default separator symbol would be "/" and could be changed in the stylesheet.

```
TYPE
 TFMTDATEmm:1..12
 TFMTDATEdd:1..31
 TFMTDATEyy:integer[4]

FIELDS
 DOBmm / "mm": TFMTDATEmm
 DOBdd / "dd": TFMTDATEdd
 DOByy / "yyyy": TFMTDATEyy
```

Figure 13: Example code for dates

3. Instrument development methods

The IS web survey instrument development process is a valuable feature of the system. In the Control Centre one codes, prepares, and test datamodels in the same well-integrated Blaise environment. The IS Internet Workshop and Server Manager can be started from the Control Centre tools menu.

Testing a web datamodel is done in two stages. First, the prepared datamodel's static pages can be previewed in the Internet Workshop, enabling rapid checking of the general look and feel of the pages. Testing the active web survey requires more steps—generating the package of specification and files needed for the web server, using Server Manager adding the package to the IIS server, and finally running the survey.

We have found that rapid testing of the structure and rules of a web survey can be enhanced significantly using prepare directives. In the example (Figure 14), when we create the conditional define “webmode” in the Control Centre (Project > Options > Conditional defines > webmode), and prepare it, the NSSE2003 modelib file is used and the LAYOUT section is processed. If we run the datamodel in the Control Centre, its web layout (Figure 15) is not usable for testing.

```
DATAMODEL PrepareDirectiveExample
{$IFDef WebMode}
  {$modelib NSSE2003}
  {$message using modelib NSSE2003}
{$else}
  {$modelib depmode}
  {$message using default modelib}
{$endif}
TYPE
  TGroupOtherClass = (Freshman,Sophomore, Junior, Senior, Other)
  TGroupOtherSchoolType = SET OF (Votec, CommJrColl,Otr4Yr, None, Other)
  TEval = (Excellent,Good,Fair,Poor)

  FIELDS
    CurrClass (1) "@BWhat is your current classification in college?@B" : TGroupOtherClass
    TypeSchl (2) "@BSince high school, which schools have you attended ...B" : TGroupOtherSchoolType
    EvalExp (6) "@BHow would you evaluate your educational experience at this institution?@B": TEval

  RULES
    CurrClass TypeSchl EvalExp

{$IFDef WebMode}
LAYOUT
 FROM CurrClass to EvalExp FIELDPANE LabelList
{$endif}
ENDMODEL
```

Figure 14: Code using prepare directives

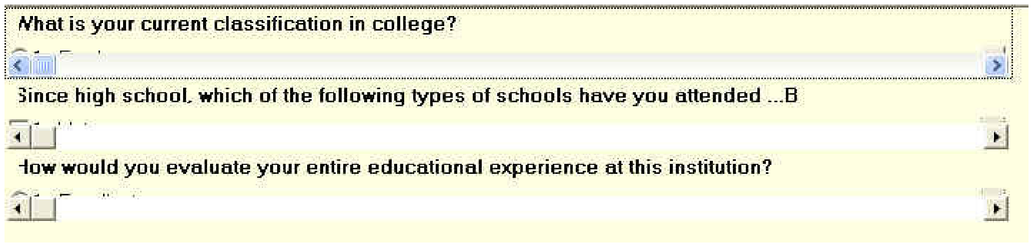
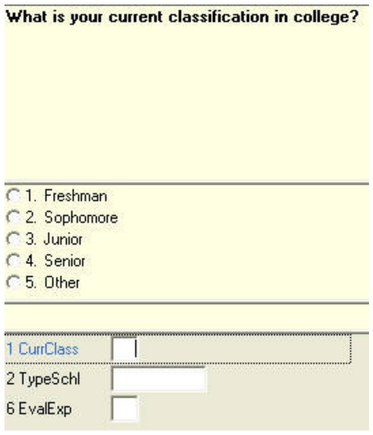

What is your current classification in college?
Since high school, which of the following types of schools have you attended ...B
How would you evaluate your entire educational experience at this institution?

Figure 15: Testing datamodel in webmode

When we change the conditional define to “depmode” (Control Centre (Project > Options > Conditional defines > depmode), and prepare the datamodel, the depmode modelib file is used and the LAYOUT section is skipped, producing a non-web look. When we run the datamodel in the Control Centre, its Windows “dep” layout (Figure 16) is usable for testing.


What is your current classification in college?

1. Freshman
2. Sophomore
3. Junior
4. Senior
5. Other

1 CurrClass
2 TypeSchl
6 EvalExp

Figure 16: Testing datamodel in depmode

We found prepare directives very useful for rapid development and testing of web surveys. It seems clear that these techniques would also be valuable for multi-mode surveys, allowing the same Blaise source to look and function quite differently in CATI and web.

4. Conclusion

With the key enhancements in Blaise 4.7 now available, we believe IS ready for serious implementations of web surveys. The XSL capability provides a complex and powerful technology for fashioning new or specialized interface elements. The architecture, user interface, and development process all appear suitable to support a wide range of household and establishment surveys.

Exploring the capabilities of Blaise IS

James O'Reilly
Westat

International Blaise Users Conference
Gatineau Quebec
September 2004

While questions remains about the effectiveness of web surveys, interest in web-based surveys has grown among researchers in government, private, and academic organizations. Some clients are pursuing exploratory efforts to learn whether the often discussed challenges--sampling, respondent cooperation, item non-response, self- rather than interviewer-administration, and the vicissitudes of the web environment--can be overcome or at least managed at a level that produces something useful. Others are interested in mixed-mode surveys, usually CATI and web. Still others want to recast existing web surveys: enhancing capabilities or increasing cost effectiveness or expanding the scope.

Westat has integrated the internet into its survey work for a number of years, including conducting web surveys. Recently growing demand and changing web research designs stimulated a company effort to examine our web survey processes and technology. One key goal is to evaluate and, where appropriate, implement web surveys using Blaise IS.

Westat's web surveys have been developed mainly using ASP, Microsoft's scripting environment for generating and processing HTML pages. ASP provides a powerful set of capabilities including easy integration with SQL Server and other relational databases. However, along with its power and flexibility, ASP is a procedurally-oriented, relatively low-level development language. This means custom programming and significant maintenance and support challenges.

Westat's ASP development team has built authoring systems to enhance our capabilities--generating much of the ASP code and reusing standard code modules. Still, it is clear that the ASP approach is problematic for some important types of surveys, for example multi-modal and more complex questionnaire designs.

Of course, all these things fit Blaise IS to a tee, in theory. Blaise is intrinsically multi-modal; it handles complexity naturally; its rules engine provide unmatched routing control; and our Blaise experience in CAPI, CATI, and CASI surveys could be applied to web efforts. But is Blaise IS ready to deliver, ready for prime-time with the architecture, scalability, usability, and extensibility required? That's what our effort was designed to address.

Our major activities were:

- User interface testing and experimentation

- Implementing XSL stylesheet extensions
- Exploring instrument development methods
- Developing a training course
- Designing and planning for server-side management and support systems

We will cover the first three of these here. Most of the work was done using IS 2.0 build 161 of 21 November 2003. As we discovered some of the limitation of IS 2.0, we awaited impatiently the release of the Blaise 4.7 beta so we could test some of the large and small enhancements. Since the beta release on July 30th we have been working intensively on version 2.1 and will report on that also.

Web interview user interface

In evaluating the user interface (UI) capabilities of IS, the core issue is that the user may often have very limited familiarity with computers, web browsers, and how to interact with them, whereas Windows Blaise UI is designed for and used mainly by trained, experienced interviewers. We focused first on elements of relatively simple web surveys of individuals, using the NSSE2003 example. We found IS 2.0 handled well scalar items in dropdown lists, enumerated, string, numeric and open text. Also it implemented nicely grouped enumerated series of questions.

The technique implementing the enumerated grouping was a critical improvement to IS. It revealed how, using naming conventions for field response types in the standard Blaise datamodel, it is possible to instruct the XSL stylesheet to change the way one or more items are rendered in the HTML page sent to the browser. We discuss this in a later section.

We found problems with the usability of complex scalar items such as date and time fields. In Windows Blaise edit masks help the user understand what components of the field to enter. Masks are not allowed in the HTML text entry control. Again extensions to the XSL stylesheet are the solution.

Other UI problems in IS 2.0 included:

- Conditional items displayed on the next page, after a trip to the server
- Limited functionality for multiple language interviews

Both of these are fixed in Blaise 4.7.

Establishment survey prototyping raised other issues, including

- Limited flexibility for controlling the table display
- Difficulty with form-like entry of information with many fields on a page and more than one field on a line desireable.

XSL stylesheet extensions

The architecture of Blaise IS provides great benefits in terms of making the Blaise system's superior features--high-level, structured language using blocks and arrays to implement complex instruments; unmatched rules engine; capable proprietary database; and other elements--available for web interviewing. To do this IS controls the process much more strictly than other web system which typically allow the use of custom JavaScript and other modules to tailor and customize the web application.

The result until recently was that using Blaise IS meant accepting a more limited and dull user interface compared to what one could expect from an ASP or other native web development approach. For some web applications, such as CATI or data editing, with trained staff as users, the tradeoff seems easy to accept. But for many customers the most important part of web survey work is the ability to conduct self-administered surveys—having study subjects do the interview themselves.

For such untrained, voluntary users with widely varying web experience a limited and simple user interface is not likely to be desireable. Expectations of many clients on the look and feel of web applications are for polished, flexible pages.

Fortunately, the IS architecture uses XSLT (Extensible Stylesheet Language for Transformations), usually referred to as XSL. XSL provides a way in IS to extend and refine key functions and interface elements in web surveys. This extensibility potential is different from and in many ways superior to that available in traditional Blaise (DLLs or the BCP).

During a Blaise web survey each time the IS server is ready to communicate with the user's browser, IS uses the API to generate an XML (Extensible Markup Language) document with the appropriate data and metadata on the fields, texts, layout, and other facets of the current situation. The XML document's rich content provides such information as the field's type which allows the stylesheet to process in various ways.

The document is processed by the MSXML parser using the interview XSL stylesheet, biHTMLWebPage.xsl, or a substitute. This process transforms the XML information into the HTML page that is sent to the user.

API > XML > MSXML + XSL > HTML

Figure 1: Blaise IS server steps

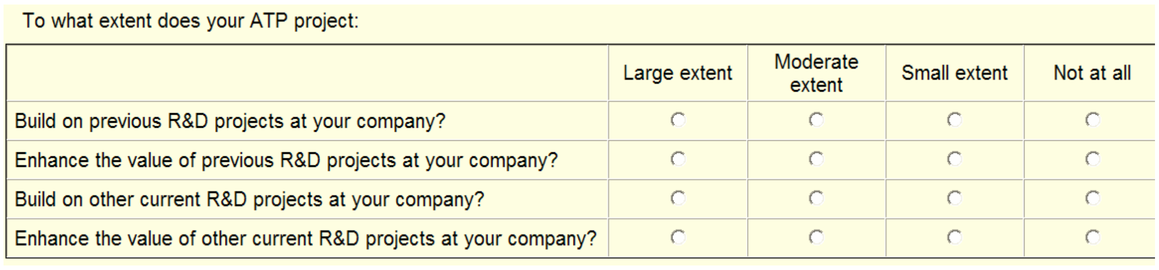
As we prototyped surveys in IS 2.0, we identified a number of user interface elements that we needed but could not render with the system as-is. Our engineers studied the system and concluded that we could generate many of the requested elements by adding functions in the interview XSL stylesheet, following the model developed by the Blaise team for displaying a series of enumerated items as rows in a table with the question text in the stub and the enumerated choices as the columns.

Here's an example. In the datamodel TYPE section a TGroupExtent is defined (Figure 2) and applied to the four fields.

```
TYPE
  TGroupExtent = (large "Large extent", Moderate "Moderate extent", Small "Small extent",
    notatall "Not at all")
FIELDS
    BUILD_PREV "Build on previous R&D projects at your company?": TGroupExtent
    ENHANCE_PREV "Enhance the value of previous R&D projects at your company?": TGroupExtent
    BUILD_CURR "Build on other current R&D projects at your company?":TGroupExtent
    ENHANCE_CURR "Enhance the value of other current R&D projects at your company?":TGroupExtent
```

Figure 2: Grouping code

The interview stylesheet has been programmed to check for the answer types of fields, looking for a series of enumerated fields with the same type name and the prefix TGroup. When a set is found the stylesheet instead of displaying the series as individual items in sequence on the page, renders them in a table (Figure 3).


To what extent does your ATP project:

	Large extent	Moderate extent	Small extent	Not at all
Build on previous R&D projects at your company?	○	○	○	○
Enhance the value of previous R&D projects at your company?	○	○	○	○
Build on other current R&D projects at your company?	○	○	○	○
Enhance the value of other current R&D projects at your company?	○	○	○	○

Figure 3: Grouped enumerated display

This model made clear that a wide range of web-standard user interface elements could be added to Blaise IS using only the standard Blaise datamodel syntax and extensions in the XSL interview stylesheet.

We then began to explore the details of implementing this capability as we prototyped instruments and found web interview interface needs that could not be met as fully as we wanted using only the standard Blaise modelib.

Scales

Using the modelib one can display Likert-type scales horizontally but not with the labels displayed above the points rather than to one side. We requested this of the Blaise IS developers which they quickly provided. In the datamodel one uses a TYPE with a prefix of TScale (Figure 4) and a corresponding function in the interview stylesheet to produce the display (Figure 5).

```
TYPE
  TScaleAmbitious = (MuchLess "Much less ambitious", Less " ", SomeLess " ",equal "Equally ambitious",
    SomeMore " ", More " ", MuchMore "Much more ambitious")
FIELDS
    PROJ_AMBITIOUS "Relative to other R&D initiatives in your industry, how ambitious would you
      say are the overall goals identified for this project?":TScaleAmbitious
```

Figure 4: Likert scale extension code

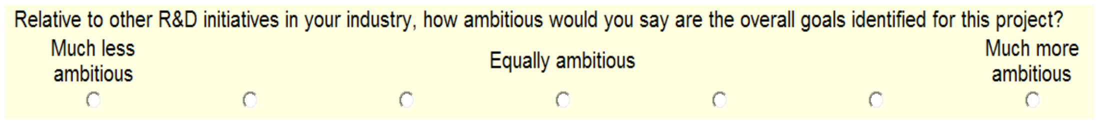

Relative to other R&D initiatives in your industry, how ambitious would you say are the overall goals identified for this project?

Figure 5: Scale display in IS

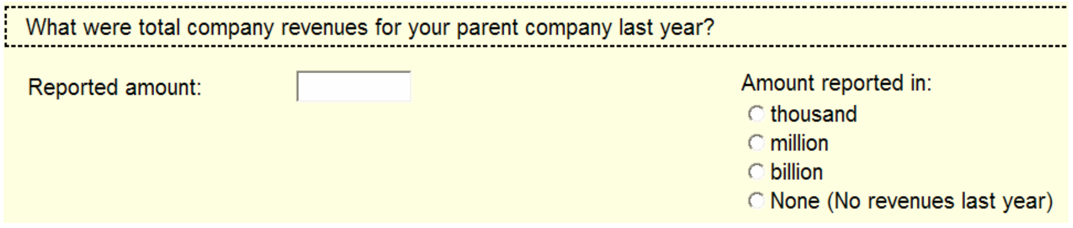
Quantity-Unit

Quantity-unit questions are important to many surveys. We need to know some quantitative information (“How much are you paid?”) and the respondent may think of the concept in terms of so much per hour, per week, per two weeks etc. Using standard Blaise with an interviewer, the two elements are different fields and the interviewer may be able to complete both based on the response received to the first part. Or a follow-on question about the unit may be required.

In self-administration this compound concept can be challenging. So we wanted the visual representation to link the two fields more clearly than a two-step dialog displayed vertically on the page. We specified that two successive fields of types prefixed with TQuanUnit would be displayed side-by-side. Our XSL specialist added this to the stylesheet. The datamodel code and display are below.

```
TYPE
  TQuanUnit_Unit = (thousand, million, billion, noRev "None (No revenues last year)")
  TQuanUnit_Quan = 0..999999
AUXFIELDS
        introrev "What were total company revenues for your parent company last year?":string[1], empty
FIELDS
       RevAmount "Reported amount:"/"Reported amount:": TQuanUnit_Quan
       RevUnit   "Amount reported in:"/"Amount reported in:": TQuanUnit_Unit
RULES
       introrev
       revamount
       revunit
```

Figure 6: Quantity-unit code


What were total company revenues for your parent company last year?

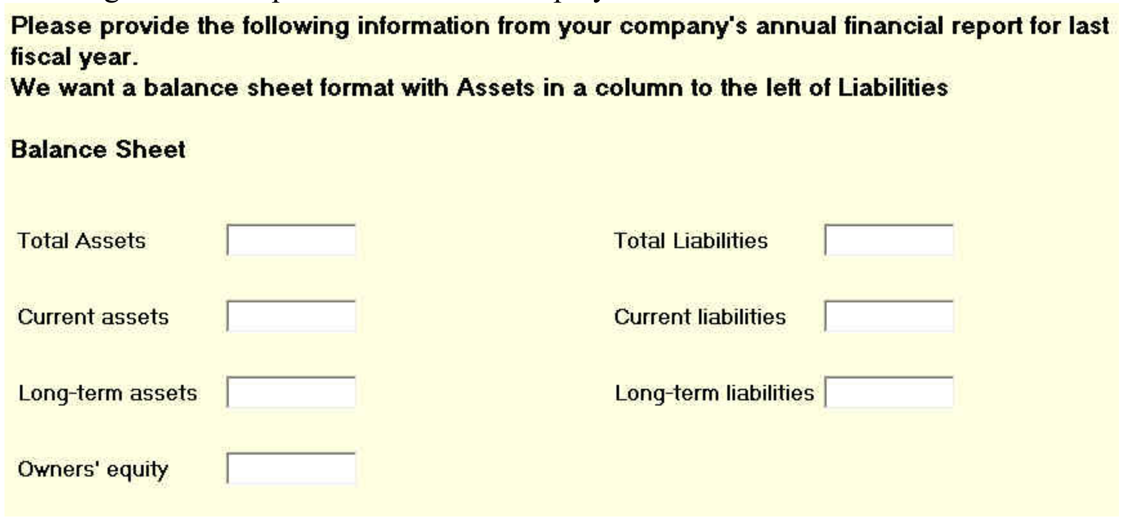
Reported amount:

Amount reported in:
- thousand
- million
- billion
- None (No revenues last year)

Figure 7: Quantity-unit display

Other user interface extensions added using the stylesheet include:

- Enabling two dual input lines to able to display assets and liabilities in two columns.

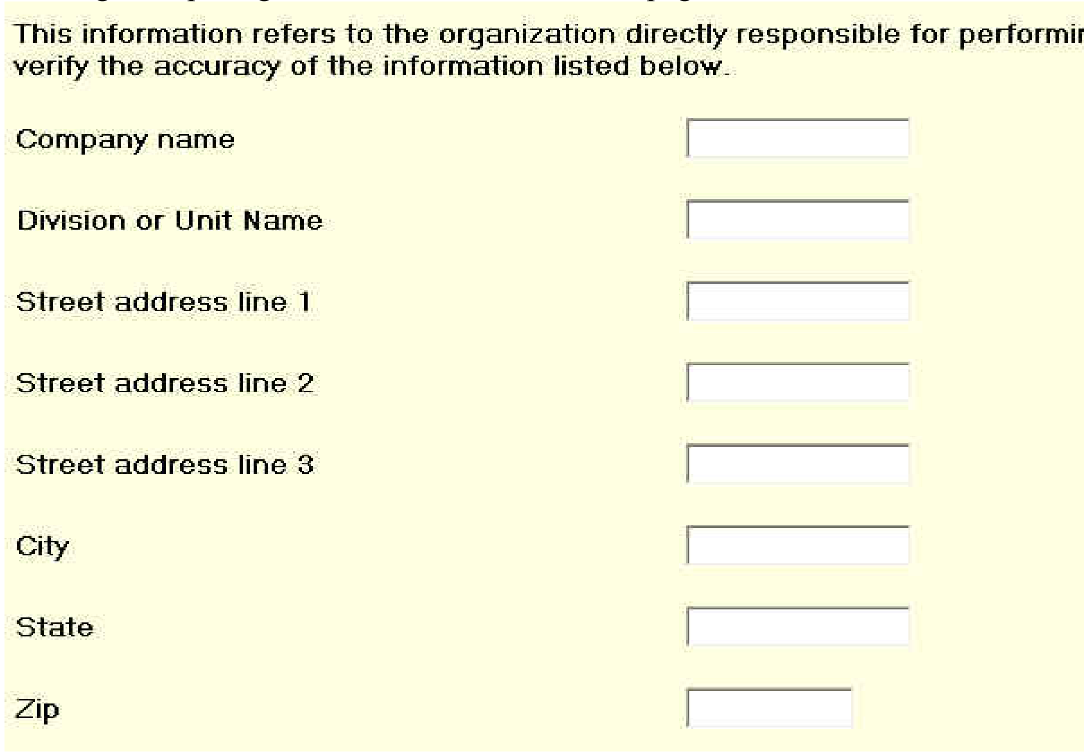


Please provide the following information from your company's annual financial report for last fiscal year.
We want a balance sheet format with Assets in a column to the left of Liabilities

Balance Sheet

Total Assets		Total Liabilities	
Current assets		Current liabilities	
Long-term assets		Long-term liabilities	
Owners' equity			

- Altering line spacing to show more elements on a page.



This information refers to the organization directly responsible for performir
verify the accuracy of the information listed below.

Company name	
Division or Unit Name	
Street address line 1	
Street address line 2	
Street address line 3	
City	
State	
Zip	

- Adjust the column widths of grouped enumerated sets so that the widths the same.
- Improved control of the width of text
- Increasing the size of the radio button and check box controls to improve ease of use
- Changing the menu and parallel tab look.

Applying custom stylesheet extensions

These extensions are added to the default IS 2.0 web interview stylesheet, biHTMLWebInterview.xsl, following XSL programming standards to prevent our references from conflicting or interacting with other functions in the stylesheet. This is done by defining the Westat namespace (Figure 8) so that all variables related to the extensions are identified by the namespace

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:westat="http//www.westat.com" version="1.0">
```

Figure 8: Westat namespace defined

To use the Westat extensions, developers start with the extended interview stylesheet. Working in the IS workshop and IS stylesheet editor the standard instrument customization is done—changing header and footer appearance, fonts, images, and the like—and saving a instrument-specific version of the extended stylesheet.

If the developer wants to change the default settings of the Westat extensions, the IS stylesheet editor can't be used. Instead a text editor or similar tool is needed to access the XSL code and enable or disable specific functions (Figure 10) or change parameters for the functions (Figure 10).

```
<!-- Westat start -->
<!-- OpenSource XSLT flow-control variables - all contributors -->
<xsl:variable name="westat:EnableQuantityUnit">true</xsl:variable>
<xsl:variable name="westat:EnableLineSpacing">true</xsl:variable>
<xsl:variable name="westat:EnableInputLineAlign">true</xsl:variable>
<xsl:variable name="westat:EnableEnumColWidth">true</xsl:variable>
<xsl:variable name="westat:EnableDualInputLine">true</xsl:variable>
<xsl:variable name="westat:EnableTabletStyle">true</xsl:variable>
<xsl:variable name="westat:EnableSpecialEffects">no</xsl:variable>
<xsl:variable name="westat:EnableMenu">true</xsl:variable>
```

Figure 9: Enabling/disabling extension functions

```
<!-- OpenSource Westat -->
<xsl:variable name="westat:DualInputLinePrefix">TDualInputLine</xsl:variable>
<xsl:variable name="westat:QUPrefix">TQuanUnit</xsl:variable>
<xsl:variable name="westat:LSCellSpacing">0</xsl:variable>
<xsl:variable name="westat:LSCellPadding">0</xsl:variable>
<xsl:variable name="westat:FieldLabelWidth">40%</xsl:variable>
<xsl:variable name="westat:FieldNameWidth">40%</xsl:variable>
<xsl:variable name="westat:InputLineWidth">60%</xsl:variable>
<xsl:variable name="westat:TabletRadioButtonWidthHeight">width:26px; height:26px</xsl:variable>
<xsl:variable name="westat:TabletCheckBoxWidthHeight">width:26px; height:26px</xsl:variable>
<xsl:variable name="westat:TabletButtonWidth">150px</xsl:variable>
<xsl:variable name="westat:TabletButtonHeight">40px</xsl:variable>
<xsl:variable name="westat:HighlightColor">cyan</xsl:variable>
<xsl:variable name="westat:ParallelTabBackgroundColor">yellowgreen</xsl:variable>
<xsl:variable name="westat:ParallelTabFontWeight">bold</xsl:variable>
<xsl:variable name="westat:ParallelTabFontFamily">arial</xsl:variable>
<xsl:variable name="westat:CurrParallelTabFontWeight">bold</xsl:variable>
<xsl:variable name="westat:CurrParallelTabColor">rgb(120,120,0)</xsl:variable>
<xsl:variable name="westat:CurrParallelTabBackgroundColor">gold</xsl:variable>
<xsl:variable name="westat:MenuButtonBackgroundColor">darkolivegreen</xsl:variable>
<xsl:variable name="westat:MenuButtonColor">yellow</xsl:variable>
<!-- Westat end -->
```

Figure 10: Extension functions parameters

Technical and organizational challenges of XSL extension

XSL stylesheets in Blaise IS open a significant avenue for extending the already powerful and broad capabilities of the out-of-the-box product. Some key features of this approach include:

- XML and XSL are industry-standard Internet technology being taught and learned widely, and not something restricted to a small niche. Courses, books, and other materials are readily available. So investing, building and applying XSL to web applications in Blaise is feasible at least for a larger organization.
- The XSL language is structured to preclude or minimize the chances of side effects in which one function or extension disrupts standard functions or other extended functions. So, following established development standards, extensions can be added with little risk of corrupting the standard Blaise XSL functions or other third-party extensions.
- Because of the richness of the XML document Blaise generates and passes, the XSL process has access to a large set of elements for its transformations and generation of the page to be sent to the browser. So the range of things that are accessible through the stylesheet is quite broad and substantial.

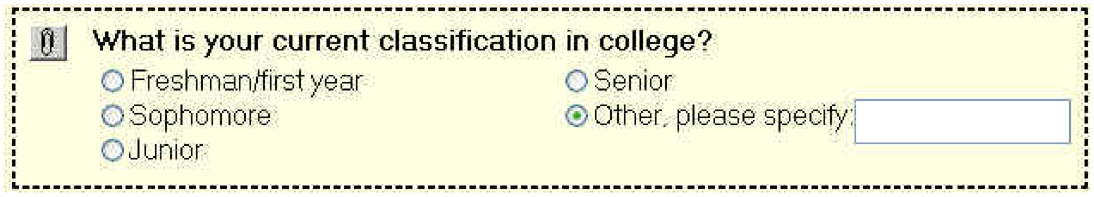
There's always a catch, of course. In this case it's that the power and potential of XSL for tailoring Blaise IS interviewing pages requires someone with serious technical skills in XML/XSL, and that person will have to become knowledgeable about the Blaise object architecture and its representation in XML.

The web interview stylesheet biHTMLWebPage.xsl for IS 2.0 has 5,335 lines and that in the 4.7 beta has 7,163 lines. Obviously, working in such a complex program requires skill and strong understanding of the details of the XSL and Blaise languages.

Does this mean that for all practical purposes extending IS web interviewing is only realistic for large organizations? This seems likely in most situations. At the same time, based on our work so far it seems that only a limited number of web survey interface elements are needed to make the system able to meet a very high percentage of web interview needs.

While the technical details of XSL programming with Blaise is far beyond the understanding of this writer, the Westat experience so far is that adding functions in the stylesheet is not as difficult and time-consuming a process for the skilled, prepared XSL developer as might be feared. Therefore we might be able to expect that the Blaise team will be willing to evolve its web interview stylesheet to add interface capabilities that users and the team agree are of general use.

Blaise 4.7 includes a powerful example of what can be done with XSL. The specify half of the other-specify is immediately activated and displayed when the "Other" choice is selected.


What is your current classification in college?

( ) Freshman/first year
( ) Sophomore
( ) Junior
( ) Senior
[✓ Other, please specify:]

Figure 11: Other-specify implementation in Blaise 4.7

No doubt the Blaise team is as excited as the IS users at the opportunity to continue to expand the usability of IS with such XSL innovations.

Wish-list of web interview capabilities extension via XSL

Here is a list of user interface extensions of general use that this writer would like to have. They are based on the work of Mick Couper, a leader in research on web interviewing and user interface issues.

Text input areas with labels linked to the text input box.
A currency symbol before or after the field.
$ [______]

A '%' symbol before or after the field.
[______]%

Specification: using a prefix of TSymLabel, fields of this prefixed type will get a text label from the field text string "[L|R|A|B][symbol]", e.g.
/"L$" {Place a "$" symbol to the left of the text entry field}
/"R%" {Place a "%" symbol to the right of the text entry field}
/"B(___) ___-_____" {Place the format string "" below the text entry field}

```
TYPE
TSymLabelPay: 1000..100000
 TSymLabelPct: 10..100

FIELDS
Salary "What do you get paid?"/"L$": TSymLabelPay
 Taxes "What percent of your pay goes to taxes?" / "R%": TSymLabelPct
```

Figure 12: Example code for labeled text entry controls

Dates in three fields with separator and labels
__ / __ / ____
mm dd   yyyy

Specification: using the prefix TFmtDate, three fields of this prefixed type will be displayed in the order asked with the field text shown below as a label. The default separator symbol would be "/" and could be changed in the stylesheet.

```
TYPE
 TFMTDATEmm:1..12
 TFMTDATEdd:1..31
 TFMTDATEyy:integer[4]

FIELDS
 DOBmm / “mm”: TFMTDATEmm
 DOBdd / “dd”: TFMTDATEdd
 DOByy / “yyyy”: TFMTDATEyy
```

Figure 13: Example code for dates

Instrument development methods

The IS web survey instrument development process is a valuable feature of the system. In the Control Centre one codes, prepares, and test datamodels in the same well-integrated Blaise environment. The IS Internet Workshop and Server Manager can be started from the Control Centre tools menu.

Testing a web datamodel is done in two stages. First, the prepared datamodel’s static pages can be previewed in the Internet Workshop, enabling rapid checking of the general look and feel of the pages. Testing the active web survey requires more steps—generating the package of specification and files needed for the web server, using Server Manager adding the package to the IIS server, and finally running the survey.

We have found that rapid testing of the structure and rules of a web survey can be enhanced significantly using prepare directives. In the example (Figure 14), when we create the conditional define “webmode” in the Control Centre (Project > Options > Conditional defines > webmode), and prepare it, the NSSE2003 modelib file is used and the LAYOUT section is processed. If we run the datamodel in the Control Centre, its web layout (Figure 15) is not usable for testing.

```
DATAMODEL PrepareDirectiveExample
{$IFDef WebMode}
   {$modelib NSSE2003}
   {$message using modelib NSSE2003}
{$else}
   {$modelib depmode}
   {$message using default modelib}
{$endif}
TYPE
   TGroupOtherClass = (Freshman,Sophomore, Junior, Senior, Other)
   TGroupOtherSchoolType = SET OF (Votec, CommJrColl,Otr4Yr, None, Other)
   TEval = (Excellent,Good,Fair,Poor)

   FIELDS
      CurrClass (1) "@BWhat is your current classification in college?@B" : TGroupOtherClass
      TypeSchl (2) "@BSince high school, which schools have you attended ...B" : TGroupOtherSchoolType
      EvalExp (6) "@BHow would you evaluate your educational experience at this institution?@B": TEval

   RULES
      CurrClass TypeSchl EvalExp

{$IFDef WebMode}
LAYOUT
  FROM CurrClass to EvalExp FIELDPANE LabelList
{$endif}
ENDMODEL
```

Figure 14: Code using prepare directives