Moving to Blaise IS

Peter Sparks and Gina-Qian Cheung, Survey Research Center, University of Michigan

1. Introduction

As the internet becomes ever more prominent the need to conduct web surveys increases. Blaise has had internet support since 4.6, but Michigan has only recently started using Blaise IS in Blaise 4.8. This paper will look at Michigan's experiences in implementing Blaise IS in their production environment, the difficulties encountered and the solutions made. Topics covered include lessons learned in following areas: written specifications, programming, server set up, data storage choices, case management, paradata capture, data export, and remaining challenges.

2. Initial Project Requirements

The initial scope of a large multi-mode survey would require use of CATI, CAPI, and self-administered web. The plan was for the survey content to change frequently as data collected and analyzed would help refine the survey. Because of the rapid and constant change aspect of the study it was decided to use one source code and prepare directives to produce the needed different datamodels needed for the different modes. The goal was to minimize the logic flow and question content discrepancies that happen when working across different instruments.

Another requirement was that data needed to be secure, and so the choice was to use Blaise's relative BOI files via Datalink and store the actual data on a secure data server for the web mode. SQL Server 2008 was chosen as the database platform. It allowed for login management as well as ease of queries against the tables, robustness of the data, and so forth.

Without a built-in case management system, our use of Blaise IS meant integration with an existing system. We chose to integrate it with SurveyTrak (distributed phone interviewing) because of the built-in reporting features. The ad hoc nature of web data collection lent itself better to CAPI than CATI.

Given the short time frame to actually implement this large and complex survey we wanted to use existing utilities and reports whenever possible. To this end we made sure the data that was captured could be exported to formats that our utilities already used.

The final requirement was to produce printed documentation that reflected the web survey itself, along with similar documentation for the other modes.

3. Challenge - Written Specifications

Needless to say, even though we had a plan for starting such a project, there were pre-requirements that we hadn't anticipated. Because of the years Michigan has spent developing and programming CATI/CAPI surveys, we have solid guidelines for how to write specifications for a survey, and how to program the survey in Blaise. However, we didn't have either for the web mode.

Hence, at the start of this process there were no written specifications, and no standards for programming for Blaise IS. The areas to be developed for these specifications included:

- Allowable controls on the screen
- Placement of the controls
- Size, color, font of the controls and text
- Background color
- Stem content and color

- Navigation behavior
- Menu items
- Buttons (placement, size, color, ...)
- Other actions (DK/RF, external lookup, consistency checks)

4. Challenge – Programming

Programming for web is very different than the effort involved for CATI/CAPI. The basic structure to the datamodel remains the same, but all the look and feel changes because of the different mode. Decisions were made tailored for the Michigan's data collection environment.

4.1 Global settings

We followed the method set by another survey package, Illume, to determine DK/RF actions. If a question is skipped then that question is assumed to have an RF response. The DK option is only allowed on particular questions and is shown explicitly. All questions are allowed to be EMPTY, whereas CATI/CAPI most questions are the opposite.

4.2 Rules changes

The programming had to be tailored to include the additional auxfields used as stem text for pages with grids. More of the items are presented in tables instead of just a single question at a time.

4.3 Question Text

The wording had to reflect a self-administered interview: layout, color, font, size, readability.

4.4 Multimedia

Multimedia could be an option (videos, sound), but the pilot survey done by Michigan, Sunflower, did not use additional multimedia.

4.5 Modelib layouts

There is a complex interaction among the information contained in the question text, what layout is going to be presented via the LAYOUT section, the actual layout itself in the modelib, how the screen in general is going to look within the menu file, and the grouping mechanism. The modelib plays a critical role in the appearance of the survey and takes more effort to use effectively than a comparable CATI/CAPI survey. The challenge was to learn how to use all these pieces efficiently.

4.6 Menu files (panels, buttons, actions)

The menu has a complexity to its arrangement, and there was a learning curve to understand how the pieces related. For example, the different panes on a page, and especially how to center text, place the navigation buttons, color the different background areas, incorporate hyperlinks and text, and call different functions from the menu.

4.7 ASP files

We needed deeper journal information (paradata), so that meant modification of the ASP files. We had to modify pages to log into the survey, log additional information to the server, and other tasks.

4.8 Stylesheet customizations

We found that we needed to modify the default stylesheets in order to gain more control over the HTML generated to make them more generic for different browsers. We found that by modifying some attributes within the stylesheet we could solve some scrolling problems, although others, such as table/cell border outlines and centering, were still a problem.

4.9 Grouping (layout, field & label selection)

The grouping editor in the Blaise Internet Specifications file proved to be a challenge because of the editor. We found that small changes in the question (code frames, renaming questions) could invalidate a group and cause the grouping work to be redone.

4.10 Critical questions

Learning about client/server communication and when a question was required took additional time during testing. On-screen fills, hiding/showing questions, and so forth require setting this for a question. This is an additional step in the programming not present in CATI/CAPI.

5. Challenge – Servers

In order to deploy a Blaise IS survey, servers are involved. Blaise IS uses a minimum of three server roles (rules, data, and web), and allows multiple rules and web servers but only one data server. The arrangements can be one server or many physical servers. The entire set of server/service roles was a challenge to set up.

5.1. Blaise IS Ports

Working with the computing support staff at Michigan, ports on different servers were opened until the right combination worked with the firewalls we had in place. The typical symptom of a port not opened is the utter lack of response when using the Internet Server Manager, or some other error when connecting. This step took much more time than expected.

In addition, these ports need to be open for communication with the web server:

- **Firewall (optional)** should have the following ports open:
    - 80 for http
    - 443 for https (Secure Socket Layer) if desired

5.2. Blaise IS Services

The challenge was to set up these servers so they could communicate with each other through the Blaise services (API, CATI, Survey Manager, Portal, Registry, Server Park, and Service Monitor). A good reference of what services are needed for each server role is found in the Blaise Help: Control Centre - Tools - Blaise Services - Type of Blaise services. The roles of Blaise services and ports are found in "Requirements for conducting surveys."

We estimated that we eventually would need two data servers for the large project, but just used one separate data server and one web/rules server for the Sunflower project.

Installation of the Blaise software was required in different locations: the developer's machine (internet installation, full), the web server (API, Survey Manager, Portal, Server Park), the data server (API, Survey Manager), and the rules server (API, Survey Manager). We did not install CATI on any of the Blaise IS servers.

Users needed enough rights to get to the web server, but once there, the services on the server handled the other communication details & access rights. Hence, the user could never have direct access to the data because it was stored on the data server, and the data was transferred via the Blaise services.

We had assumed that we would need to create our own relative BOI file -- but Blaise (4.8.1) automatically creates the directory structure and copies the appropriate files to the data server, in addition to the rules server. This took the guesswork out of the installation.

6. Challenge – Data Storage

The most obvious choice for a data storage format for Blaise IS would simply be to use a BDB format. It would allow instant access to the data through use of existing utilities, and with use of a data server and the implied relative BOI would remain secure behind a firewall. However, there are additional features of SQL that were attractive that influence the choice.

6.1 Data Server

A separate data server was used to store the Blaise data apart from the web/rules server, and required different user logins & ports opened.

6.2 Data Roots

Blaise allows definitions of additional directories to be stored in a list of data roots to be used with BOI files. At the beginning of the process the need of data roots were unknown and somewhat confusing.

6.3 SQL backend

In preparation for the large study and knowing there was going to be heavy use of the database from both users and analysts, Datalink with SQL was a logical choice over a BDB. Blaise inherently uses datalink, and so setting up a BOI file for data use means the most flexibility in working with the data.

Using a SQL database also allowed us to use three different "databases" within the database: main interview data, working data, and paradata. This kept management of the files tidy and secure.

6.4 BOI data structure (Stream, Flat No Blocks) - Datalink

We had to examine the different layouts available and choose a BOI format that worked for our needs. Two different formats to the SQL database were chosen for this study: stream and "flat, no blocks." The former was used for the main and working databases, while the latter was used for the paradata. The stream format was chosen for its compactness and speed. It cannot be queried directly via SQL, but through BOI files it can be easily exported to another format. The flat format was used for the paradata to allow it to be queried as needed, and in fact the tables created were used to store timing information back to the sample management system, SurveyTrak.

6.5 Minimize number of tables (Main, Working, Paradata)

To minimize the amount of overhead, each study published in Blaise IS was assigned a database within SQL. This allowed all the associated tables (main, working, and paradata) to be placed in one area. Although generic tables could have been chosen, we chose for this test to let the default configuration remain.

6.6 Username/password login to tables

A study-specific login (username, password) to the SQL database was created. This provided an extra level of security to the survey data. The original BOI file is password protected (via Blaise) to prevent revealing the database login.

6.7 Datamodel changes/migration

We made plans for migrating the data during production if needed. From our testing it seemed like the best plan would involve the following:

1. Update the datamodels
2. Create new BOI files that match the updated datamodels
3. Suspend the survey
4. Run an OLEDB (old datamodel) to Blaise (old datamodel) to save the data

5. Delete the survey
6. Install the new survey
7. Run a Blaise (old datamodel) to OLEDB (new datamodel)
8. Start the new survey

6.8 Database "locks" – hangs

During development we encountered locks on the database. Although we never tracked down exactly the cause, this seemed to be caused when we deleted/reinstalled a survey through the Survey Manager.

We found out that most often the process responsible for causing the lock was the Blaise API service found on the data server. We tried a variety of ways to release the deadlock that stopped the survey from responding, including rebooting the machine that was hosting the data server and reinstalling Blaise. We found that by stopping and restarting (and sometimes killing and starting) the service we could regain control. A simple solution of just waiting sometimes worked. We did find that once production began and we stopped uninstalling the survey we experienced no other locks.

7. Challenge – Case Management

Blaise IS provides a great base for creating web surveys, but does not provide the management piece. This includes the tracking and management of sample, status reports, and holding/releasing/preloading cases into the system, as well as sending emails to respondents.

7.1 Existing system (CAPI) integration

SurveyTrak is the sample management system used for our decentralized interviewing system. It has, throughout the years, had many management tools built into the system. We decided to use existing features in SurveyTrak and add a few more functions outside of SurveyTrak.

7.2 Email jobs

Email is an important way to contact the respondents, but unfortunately Blaise IS does not provide a system to do this. However, a new module was written to handle email jobs (see the conference paper, "Blaise IS Sample Management."[1]).

7.3 Suspend and resume interviews

The respondent was required to log into the survey (after being provided the required information in an email). Upon resuming an interview the respondent was taken back to the place they last suspended.

7.4 Reports

Blaise IS's Internet Manager provides very basic reports on the number of completes and the number of times the survey has been accessed, but other reports needed to be customized. These were created in part from the email utility, and others from integrating with SurveyTrak.

7.5 Daily updates to SurveyTrak

Since SurveyTrak was used to supply sample to the study and run reports, daily updates of paradata information from the paradata table to SurveyTrak was needed. Information stored to SurveyTrak included the time spent by the respondent in the interview, the last question answered, and the number of resumes.

8. Challenge – Paradata

We needed detailed journal information -- much more than the default Blaise IS journal:

Field Name	Type	Description
Date	DATETYPE	Date of the event
Time	TIMETYPE	Time of the event
PrimaryKeyValue	STRING	Primary key value of the interview

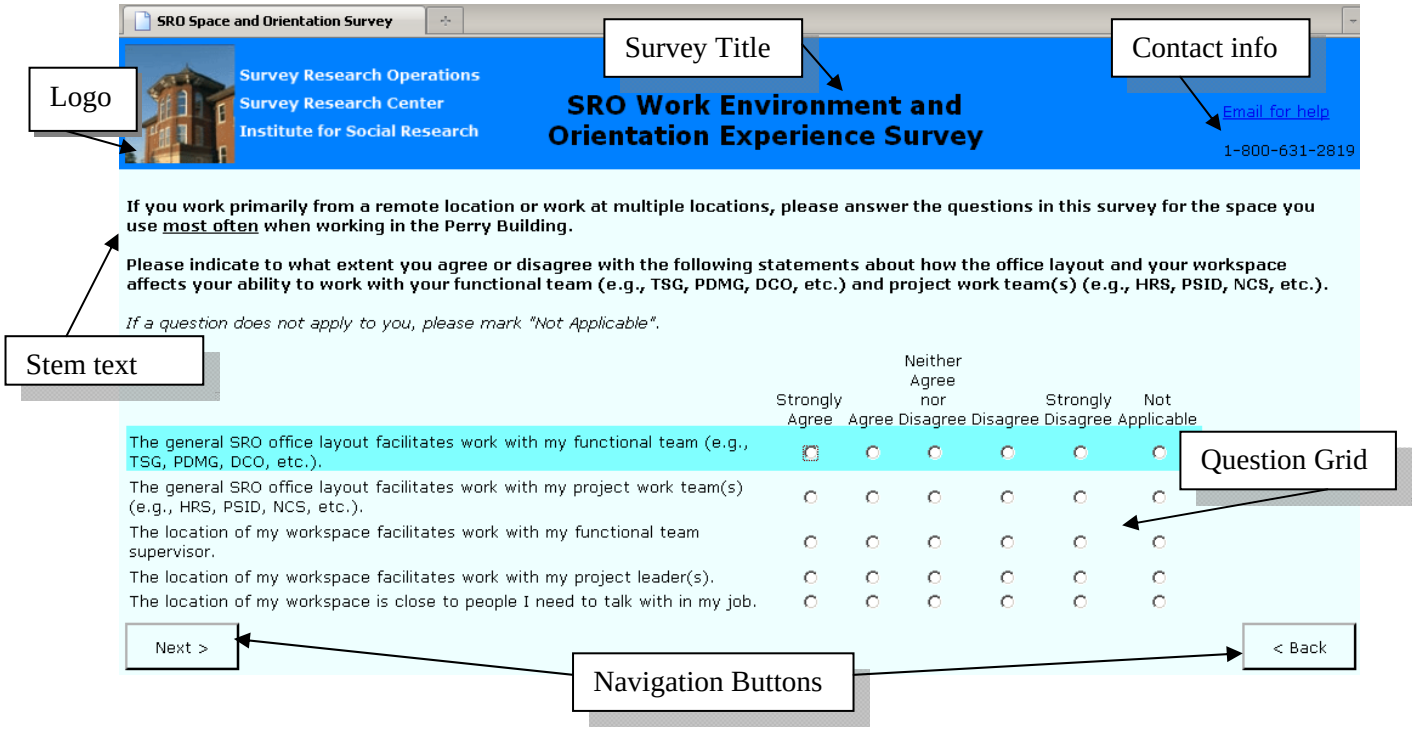
-- Blaise Help - Monitoring - Web journal for programmers

The datamodel that describes the journal can be extended within the datamodel, but we needed to store information outside the interview. These fields were populated using the menu file via javascript and ASP code. This is detailed in the conference paper, "Blaise IS Paradata."[2] The data defined in the paradata (journal) datamodel are stored in an SQL table using the Datalink "flat, no blocks" format.

9. Solution - Written Specifications

Although it would have been nice to have a lengthy time to research all the best web designs and implement them fully within Blaise IS, time was always of the essence. That is, the "specs" were tailored to the current need and limitations within Blaise IS as the instrument was being developed. There was difficulty in determining the best method of describing the layout of a screen, and eventually a simple set of markers in the spec document was used.

The markers described the start of a screen, and then the start of a grid or other structure. Any text that appeared before the start of the grid would become "stem" text that required additional programming.



General layout elements, such as the logo, survey title, help reference, button navigation, background color, highlight color, font size and font type, and so forth were initially chosen, but revised as the project went forwards.

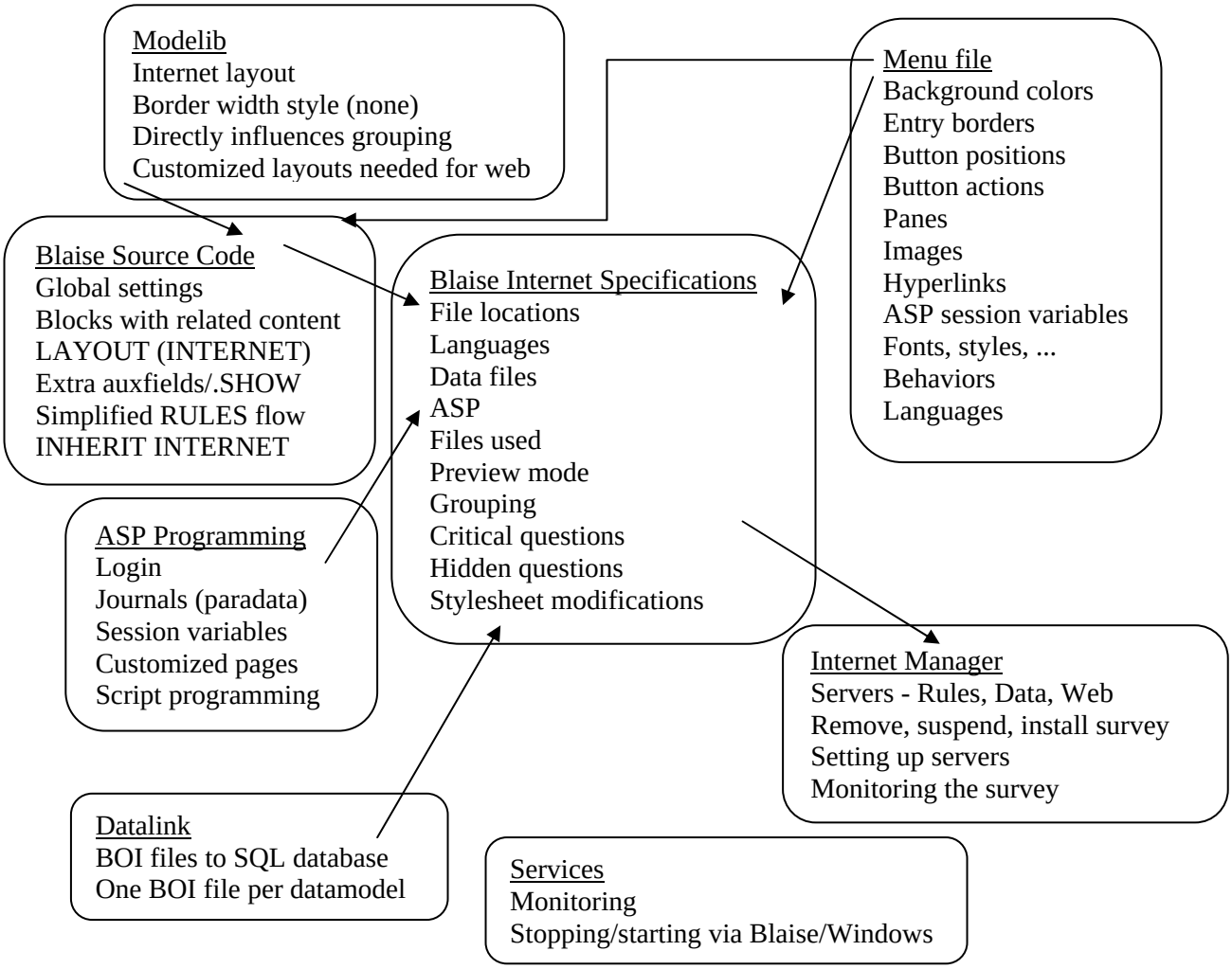
Because web surveys are self-administered, close attention was paid to making the question text distinct from the answer categories. In the above screenshot, the question text is in bold, instructions specific to this page to the respondent are in italics, and the question category topics are in a normal font.

There are areas that have not been decided. These include the use of tables, lookup lists, and other user controls (drop down lists, date and time pickers, custom controls, if ever to show DK/RF values, and so forth). Therefore we expect the specifications to change as we become more versed in web studies.

10. Solution – Programming

Necessity is the mother of invention -- and that's no tall tale for entry into web survey programming. The complexity of going into this area is about the same level of complexity as learning the CATI call scheduler and setting up related systems.

A big necessity was simply learning about the different areas and how they related to one another. In CATI this process seems fairly straight-forward, with the complexity coming from the INHERIT CATI command and the call scheduler parameters (ignoring customizations to the system via manipula). However, in Blaise IS, the diagram becomes more complex because there are so many interrelated parts.



10.1 Appearance Specs Required

Programming in the Blaise instrument is influenced by the layout of the web page, so realistically the general layout of the survey needs to be specified first (and programmed in the menu file). The specific

design of each web page also needs to be determined early to help determine any "stem" text for the page, as well as any needs for grouping questions (grid, table, amount/per, or other format).

10.2 "Stem" text

In order for question text to appear before a grid, an auxfield is created. It is placed in the rules with a .SHOW attribute, and a layout that displays the question text only (via the modelib). In this example, the questions in the grid are grouped together via the Blaise IS grouping editor and via the modelib, and the NEWPAGE command is used start a new web page.

```
FIELDS
   xA1 "@/@T@BIf you work primarily from a remote location or work at multiple locations,
       please answer the questions in this survey for the space you use @Umost often@U
       when working in the Perry Building.@B@T

       @/@/@T@BPlease indicate to what extent you agree or disagree with the following
       statements about how the office layout and your workspace affects your
       ability to work with your functional team (e.g., TSG, PDMG, DCO, etc.)
       and project work team(s) (e.g., HRS, PSID, NCS, etc.).@B@T

       @/@/@IIf a question does not apply to you, please mark ""Not Applicable"".@I" :

      TContinue

RULES
   xA1.SHOW
   A1a
   A1b
   A1c
   A1d
   A1e

LAYOUT (INTERNET)
   BEFORE xA1 NEWPAGE
   AT xA1 FIELDPANE QuestionTextOnly
   FROM A1a TO A1e FIELDPANE GroupTableQuestionTextLeft
```

10.3 Menu modifications

One frustrating thing about working with the panel and placement of controls on the menus is that the dialogs work with pixels, but realistically it's a proportion of available space (height and width) on a web page. Placement of an item is relative to the widths of all the items.

We found that the background color of an object will carry through to the controls below it. So if a panel is colored green, then all the things on the panel also get green unless it's reset.

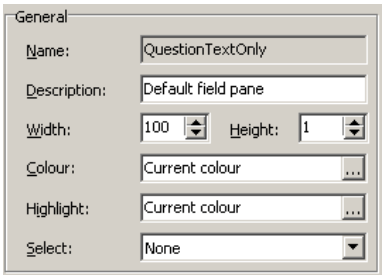
We also recently added a variable to the menu for our internal CAI Testing Tool utility to have it work with Blaise IS surveys. The use of this variable may be discussed in the paper, "BlaiseIS Paradata."[2] An earlier paper on the CTT was given in "CTT: CAI Testing Tool."[3]

10.4 Variables

Programming for a Blaise IS web page is more intensive than for CATI, and will carry more auxfields. It's recommended that programmers use a standard naming convention for the additional auxfields, such as xFieldname when its purpose is introduction text to a page.

10.5 Layouts

Blaise Internet requires a modelib specific for that mode. Within that mode there will be a number of default field panes for internet. We found it useful to customize a few additional fieldpanes.



One setting we universally did was to remove the dashed highlight around the question text. This was accomplished by choosing a fieldpane, then selecting "None" for the question select in the General group.

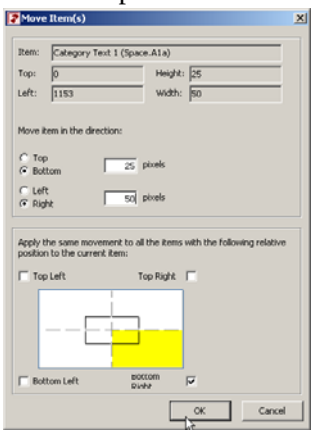
10.6 ASP - login & paradata

We modified the active server pages to provide a different login, and also to store survey data from the interview back to the SurveyTrak database.

10.7 Grouping

Grouping in Blaise IS is a huge challenge, and becoming familiar with the editor does speed up the process a little. Given the fragility of grouping (a change in the datamodel could make the programmer redo the grouping) this should be one of the very last things done to a web survey. Make sure the survey is solid, the logic and fills work, and the pages are correct, and then finally do the grouping.

We learned that what appears under the grouping editor is influenced by what is shown in the modelib and what fields are in the datamodel. To work with a group in the editor, follow these steps:



1. Create the appropriate layouts in the modelib.
2. Make sure all the elements you want in the grouping editor are visible in the modelib (or not)
3. Make sure the layout in the datamodel is correct
4. Prepare the datamodel
5. Define the group
6. Edit the group

We found saving our grouping often was beneficial.

Like the panel and positioning of controls in the menu editor, the placement of items within a web page in the grouping editor is really based upon percents. So if the page width is 2000 pixel, and a control is 500 wide (and your theoretical web page width is 3000), the control will fill 1/4 of the web page, not 1/6. The elements in the group will also expand to fill the data up to a point. So for editing all the groups eventually we came up with the following method:

- Set the global height and width to 1.
- Set all border margins to 2.
- Set all category items to 50 wide.
- Set all question text to 1150 wide.
- Space rows of items 25 apart (each is 25 high)

- Use the "move items" (right-click, choose "move items") to move a cluster of items at the same time. In this picture, everything to the right and below will be moved to the right 50 pixels and down 25 pixels.
- Be patient when moving items & undoing the moving of items (it can take time).
- Do use the multi-select with the control key. Change the attributes of multiple items at the same time after using the multi select.

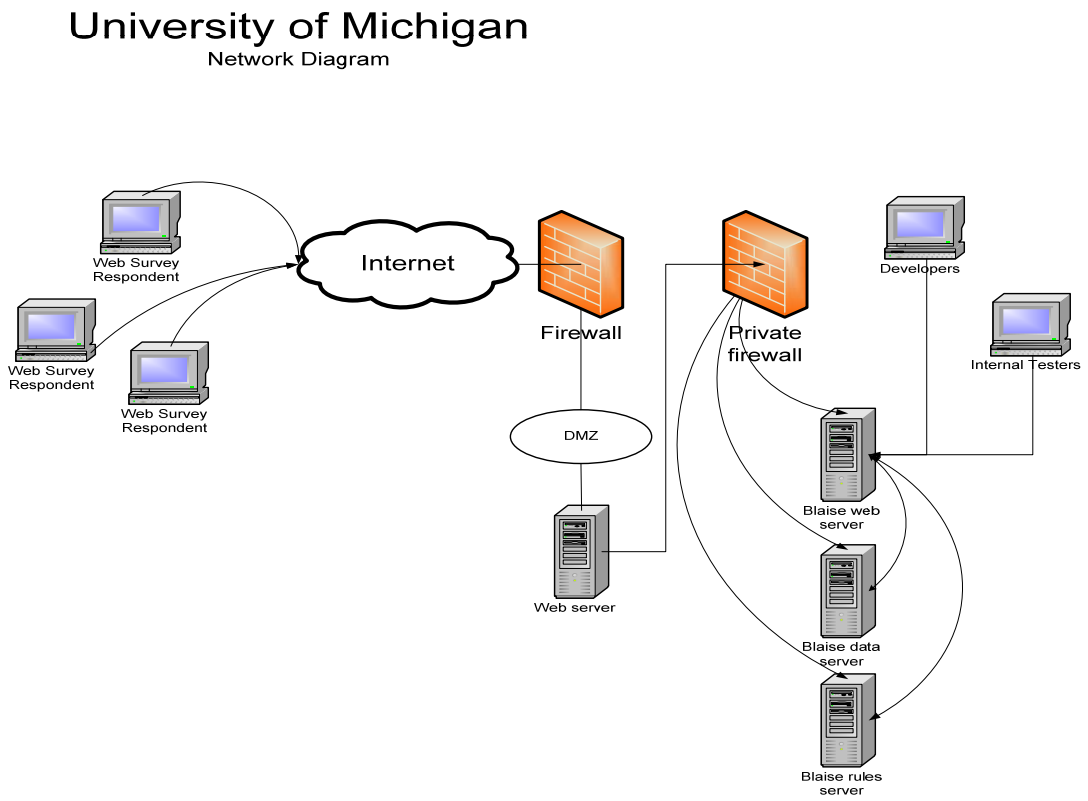
10.8 Stylesheets & HTML

We found that the stylesheets included in Blaise IS work well for Microsoft's Internet Explorer, but generated html code that gave odd results in other browsers. Notably, in Firefox we found that not all the gridlines would appear, in Chrome all the cells within the tables were centered, and similar problems in other browsers. We did not fix this problem due to time constraints, but conjecture it is due to the rendering of the html page from the stylesheet.

We also discovered that horizontal and vertical scroll bars would sometimes appear even though there were no content to be shown. After investigating, we found that the generated html page consists of a number of tables within each other, and the problem was caused by a calculated border width defined for some of those tables. Replacing the calculation with a constant "1" solved the problem.

11. Solution – Servers

The intent was to have three physical servers, and add in additional rules servers as needed. However, we ended up with three virtual servers that acted the same. Since our first project had such a small sample we never really tested the load on the server itself, but we also did not notice any problems for running the survey.



In preparation for the large survey most personnel were being moved behind an additional firewall within the Survey Research Operations. This caused a problem with just being able to access the web server: it was behind the additional firewall, and users who wanted to test from outside the building could not. The resolution was to have the users log in to the servers within SRO via VPN, and then they could access the web site.

The computing support staff made sure that the communications that Blaise IS needed to talk among the servers via the ports were done correctly, and the appropriate ports were opened.

It was confusing at first to learn where the files were placed when the Internet Manager deployed a survey. We used Blaise 4.8.1 for our first survey data collection, and hadn't known that the when a survey is deployed it gets copied to multiple servers, and found it best to let Blaise handle all the files via the package (BIS) definition. We also found that Blaise 4.81 made relative BOI files automatically for use with the data server we had set up.

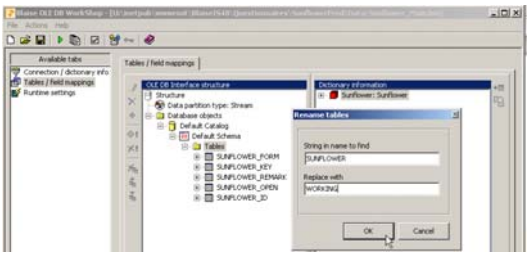
12. Outcome – Data Storage

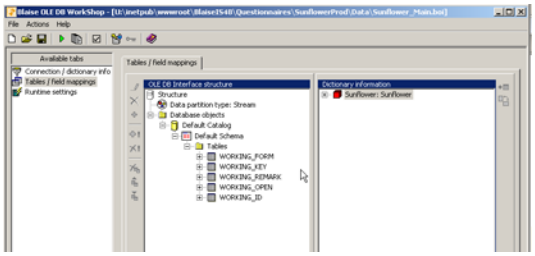
We had to find out which was better: Blaise BDB, SQL storing Blaise data, or both?

Not surprisingly, Blaise BDBs are very handy when using existing utilities. The processes are known for retrieving data, running reports, and maintaining the databases. The downsides are typical -- multiple files per datamodel, file locking problems and speed of access in shared mode, and the possibility of an "oops" moment by deleting a file/moving a file using the file explorer. Placing the data in SQL tables did protect against the downsides (not as likely to accidently delete data/corrupt the data, and won't restrict access speed to the data), as well as provide more security to the data.

The paradata (journal) information was stored in a "flat, no blocks" format that made it very easy to run SQL queries and work with the data using stored procedures and C# utilities. The questionnaire data, stored in the stream format, was used to minimize storage and increase access speed for the respondent. So we had to still export the interview data into a BDB for analysis, but were still able to run basic queries (i.e., number of completes/partials) against the SQL data.

We did find it easy to store all the related survey databases (main, working, and paradata) in the same SQL database without the generic format. We created the tables by making the BOI files in the OLEDB Workshop. After creating the main BOI, we edited a copy of the BOI to point to the Working database by renaming the tables in the main. This created a new set of tables in the SQL database that referred to the same main datamodel.





We never did fully resolve the problem of having the data server hang, but by the same token we did not post a new survey during the data collection period. Therefore, the "survey locks" is still an outstanding issue.

13. Solution – Case Management

A basic case management system was developed for the web mode that extended the functionality of SurveyTrak. A list of email addresses and status were stored in a database, and an email module managed the sending of invitations and reminders. Once the respondent began or finished an interview, that information, along with a summary of the paradata, was stored in the SurveyTrak database. Managers could run existing reports from SurveyTrak, or additional queries against the Blaise IS tables stored in the SQL database. The email module was a major addition to the system, and the discussion of that system can be found in reference (1).

There were relatively few problems in developing the stored procedures to store the Blaise IS paradata and case status to the SurveyTrak tables. Those problems were more of the technical communication methods needed rather than any system design issues.

14. Solution – Paradata

The details of the paradata (journal) information can be found in reference (2). The benefit of the paradata was that it enabled managers and researchers to analyze respondent behavior down to the field level on a page, and included their navigation actions. This is more detailed than the default journal file.

To achieve this required new standard, this included disabling the default journal, modifications to the ASP code, a new standard datamodel for storing the data, and new SQL tables and stored procedures to summarize and transfer the data.

This doesn't have any visible affect on the respondent (i.e., no slowing down of the web page), and the setup overhead is minimal now that the standards have been defined. The development did take some effort.

15. Lessons learned

We learned the following.

- *Standards before implementation*
  This is well known, but bears repeating. Without clear standards before implementation starts, there will be much redoing and discarding of work, and ultimately will take more time and cause more problems than tackling the standards up front.

  When time is an issue, it's better to still agree to something and have it written down even though that standard may not be perfect. The point is that there is a common document that everyone uses, and the tendency is that without that document there are no standards.

- *New systems require lead time*
  One reason we used virtual servers is because they were fast to implement using existing hardware and we didn't have to order new servers. The downfall is that there are multiple processes running on the same physical server, thus, overall, the virtual server cannot handle the same load as a physical server.

  Even so, setting up the servers required time to install software, assign users, roles, and groups, and to make sure everything works. Technical glitches can easily delay the start of server use until issues are resolved.

The case management system was actually designed and implemented while the survey programming was being done. The last parts implemented were the data export and management reports for the new system.

- *Learning curve*
  Stepping into the Blaise IS world took a great deal of time because of the many components, and required a broad set of skills to master. It was a task best handled a little at a time, but with a big picture overview to help guide the process. A training class that gets down to the littlest details would be helpful.

- *Each area has its own challenges*
  This includes the datamodel source code, modelib, menu, internet specifications, BOI files, ASP, stylesheets, SQL, SurveyTrak, services, servers, file locations, ports, and user rights. Each of these had its own challenges, and it was best to concentrate on learning one part at a time.

16. Summary

The move into Blaise IS has not been without its challenges, but the journey has been worthwhile. Blaise IS offers a rich environment to develop and deploy web surveys, but needs additional systems to supplement the features that are in the internet package. A great deal of customization can be accomplished within the system.

Michigan has greatly expanded their capabilities in the area of web surveys in Blaise, and is ready to conduct additional surveys more efficiently. Future directions include resolving some outstanding issues that were discovered during this first foray: defining archiving procedures, correcting browser incompatibilities, providing more SQL stored procedures for accessing the data, and updating testing tool and the case management system and reports.

References

[1]Hueichun Peng and Lisa Wood, BlaiseIS Sample Management, IBUC 2010
[2]Jason Ostergren and Youhong Liu, BlaiseIS Paradata, IBUC 2010
[3]Mary Dascola, Genise Pattulo and Jeff Smith, CTT: CAI Testing Tool, IBUC 2007