Making Blaise 2.5 Do Things It Was Never Meant to Do

James M. O'Reilly, Research Triangle Institute, USA

Abstract

Blaise 2.5's fundamental architecture allows direct access to the most elementary interviewing functions and processes. This allows one to extend 2.5's functionality in important new ways. The paper discusses the Blaise 2.5 architecture and describes how, for a methodology study, we added a keystoke capture mechanism. The system saves every keystroke entered during an interview, both human-readable and special PC keystrokes. In addition to the keystrokes the system save a time stamp to the hundredth of a second each time the [Enter] key is pressed. The keystroke stream is saved to a separate data file periodically.

1. Introduction

Blaise 2.5 has limitations for some applications, and it lacks many of the impressive powers of Blaise III. Still, 2.5 remains widely used. Some are reluctant to change proven legacy applications. Others continue to develop new 2.5 applications because Blaise 2.5 is adequate to the task, because it may more efficient to build on the already established staff knowledge and experience, and other reasons.

Blaise 2.5 also possesses another "secret" feature which may make it useful in ways that Blaise III is less capable. Blaise 2.5's fundamental architecture allows direct access to the most elementary interviewing functions and processes, allowing one to extend 2.5's functionality in important new ways. In the Blaise 2.5 system various standard modules of Turbo Pascal code can be changed by the programmer so that when the application is parsed and compiled entirely new functions can be implemented.

The paper discusses the Blaise 2.5 architecture and describes how, for a methodology study, we added a keystoke capture mechanism. The system saves every keystroke entered during an interview, both human-readable and special PC keystrokes. In addition to the keystrokes the system save a time stamp to the hundredth of a second each time the [Enter] key is pressed. The keystroke stream is saved to a separate data file periodically.

2. Why abandoning Version 2.5 may not be wise

Blaise III is clearly the most powerful, comprehensive, and elegant computer-assisted interviewing system available today. It is a major leap forward from its predecessor Blaise 2.5, not to mention the competitive products which were, and are, inferior to Version 2.5 in many significant ways.

Blaise III has expanded dramatically the framework for computerized interviewing system to become the first true “survey processing system”. Blaise III’s object-oriented architecture, advanced meta data framework, and integration of the interviewing/data entry process with the surrounding components of survey processing provide survey systems designers and developers with a much richer, more powerful, and more flexible environment.

Blaise III is clearly the system of choice for serious DOS-based survey processing projects. At the same time it is important to point out that there may be situations where Blaise III is not the superior choice over Blaise 2.5. Among these might be :

- where existing Blaise 2.5 applications are performing properly,
- where “legacy” DOS hardware must be used (Blaise III can be quite slow on older x86 processors and on machines no extra random access memory (RAM)), and
- similar situations where the significant investment in software licenses, learning the new system, and retooling existing processes cannot be justified yet.

3. Limitations of 2.5 and user work-arounds

Blaise 2.5, as with any application, was developed to meet the realities of the computer environment of the time it was designed and built. This meant the world of DOS, 640k of programmable memory, Turbo Pascal, character screens, and no doubt many other constraints.

The resulting system has limitations which, as clients have expanded the scope of what they want done in computerized surveys, have become obstacles. Among these are, for a single instrument: a maximum of 2,000 items, 64k limit on the amount of question text, 64k limit in the size of enumerated answer sets, only one language can be used, and others.

One of the great frustrations of many computer experts and gurus is that, in their view, MS-DOS is a profoundly flawed, out-of-date, makeshift crime against all known principles of operating systems design. And it should have been strangled at birth, or at least buried permanently ten years ago. While the technical logic of the gurus is not seriously questioned, the market and the ingenuity of users seeking to solve their immediate problems with the tools and resources available have failed to pay attention to the dicta of the experts.

Countless work-arounds, adaptations, extensions, and other ad hoc strategies have been found to make MS-DOS do things it was never thought to be capable of doing. A similar situation has occurred among Blaise 2.5 users. As limits have been reached, creative solutions have been found to extend the system in new ways. A number of these strategies have been reported in Blaise User Group meetings and in the user group's newsletter.

One example is the 2,000 question limit. A number of studies have successfully bypassed this restriction by converting their instrument into two or more separate sequential Blaise instrument in which the later sections read data from the prior ones using the valuable Blaise 2.5 external file access interface. At Research Triangle Institute we have recently built a survey made up of 13 separate Blaise 2.5. Each respondent actually encountered eight instruments. The first, seventh and eighth modules were done by all, while modules two to seven had two versions to respondents were randomly assigned. A similar multiple Blaise 2.5 architecture was used successfully in the National Survey of Family Growth (Duffer and Moser, 1996; O'Reilly, 1993).

The separate Blaise instruments are executed by a driver process—either a DOS batch file or for more complex requirements a custom application in FoxPro, C, or another language. Of course, work arounds have their costs in terms of the extra effort to structure all the components to work properly together. And Blaise III eliminates the need for many of these work arounds, particularly those arising from Blaise 2.5 memory limitations. Blaise III used DOS extended memory so that most or all of the previous limitations not longer exist.

4. Version 2.5's peculiar architecture

The fundamental strengths of Blaise 2.5 and Blaise III derive from the creative and comprehensive design implemented by the Blaise team at Statistics Netherlands. Each of these two very different systems was designed to incorporate the most advanced and appropriate applications design principles of its day to a clearly conceived target of survey data processing. The power of this approach and the clear thinking, talented technical skills, and overall leadership that has supported it is, to this long-time user, most impressive.

One of Blaise 2.5's fundamental strengths is its speed of execution, even on 286-class PCs running on a floppy disk. When Blaise 2 was conceived execution speed as a key issue for applications. It was not something one could take for granted, as designers do today when they assume that users will have a Pentium-speed processor, 16 megabytes of memory and a very fast and large hard disk.

Blaise 2.5's execution speed is a function of the fact that a Blaise 2.5 application is a DOS executable process. Other systems then and now, including Blaise III, use an standard executable engine which interprets the application meta information and executes the application. Interpreted systems are significantly slower that executables. But, of course, this usually matters only on older, less powerful hardware.

Blaise 2.5's architecture has these components :

1. A parser which reads the Blaise application code, checks for syntax errors and, for an application with no errors, generates a series a Turbo Pascal source files and data files with the application's execution logic, data requirements, and question definitions.

2. A set of standard Turbo Pascal source files which handle various fixed elements of an application—getting keyboard input, managing screen display, handling data, executing the question flow.

3. The Turbo or Borland Pascal compiler which builds the DOS executable system from the application-specific and standard Pascal source files.

5. Extensibility and Version 2.5

The ability to extend a software system to implement specialized functions not in the core development system is an important goal of systems designers especially in recent years. In the Windows environment and web application development a major feature is the ability to utilize flexible and powerful objects using such technology as DLLs, ActiveX, Java, OpenDoc, COBRA. Blaise III includes a DLL interface to provide a capability to extend applications with specialized tools. This DLL interface offers far greater speed and integration with the standard Blaise application that those offered by competitive systems.

Blaise 2.5's open architecture of Pascal source files offers a non-standard, unsupported, and unorthodox avenue for extending survey applications. By altering these Pascal units and procedures one can change how a Blaise 2.5 application works in quite fundamental ways. In fact, this provides the 'backdoor' approach to extensibility with at least some functionality that an application developer in Blaise III cannot do either as well or possibly at all.

6. Caution

It should be clear that anyone doing this type of reverse engineering of Blaise 2.5 is doing so at his or her own risk. Statistics Netherlands is not responsible for anything that might result, and does not support developers who are, uninvited, making changing to the core processes of the system.

Similarly, neither the author nor Research Triangle Institute take any responsibility for any problems which others may encounter in manipulating Blaise 2.5 as described in this paper. Some valuable functionality can be created using these methods. But it should only be attempted by persons with the requisite programming skills and persons who take full responsibility for any results.

7. Reverse engineering Blaise 2.5

The fundamental approach used is to study the standard Pascal source files to understand what the system is doing and where. This approach is aided by fact that the Pascal is a strongly-typed, highly structured language which was designed originally for teaching and learning programming. As a result the code is more “wordy” and amenable to someone else puzzling through the purpose and details of the code than C and other languages with dense and cryptic syntax.

Another aid in learning how the interior of Blaise 2.5 works is the programmers source file comments. While the source code is not extensively commented, many helpful comments are included. Some are in English and some Dutch. Although we have not done it yet, at some point one would find useful a Dutch-English (or French, etc.) dictionary.

The table below shows the names of the Blaise 2.5 Pascal standard source files.

APPOT25.PAS	HELPX25.PAS	RAUTLX25.PAS
BOMENO25.PAS	INCONO25.PAS	RDCHRX25.PAS
CALLT25.PAS	INFRMD25.PAS	REMRKX25.PAS
CHECKD25.PAS	INITD25.PAS	SCRNX25.PAS
CHECKP25.PAS	INITP25.PAS	SHOWQD25.PAS
CHSTRX25.PAS	INSESD25.PAS	SHOWQP25.PAS
CHUTLD25.PAS	LIPSP25.PAS	STARTP25.PAS
CHUTLP25.PAS	LOCKX25.PAS	STATT25.PAS
CODERX25.PAS	LSTUTX25.PAS	TEXTD25.PAS
CODUTL25.PAS	MARKP25.PAS	TEXTP25.PAS
DATAX25.PAS	MEMX25.PAS	TOOLSX25.PAS
DIALT25.PAS	OVLD25.PAS	UPDATX25.PAS
DIVET25.PAS	OVLP25.PAS	USERPX25.PAS
EDITX25.PAS	PAGEX25.PAS	VIEWRX25.PAS
EDSTRX25.PAS	PARSX25.PAS	WNDWX25.PAS
FILERX25.PAS	QUTLX25.PAS	

From the DOS file name one can make an initial guess about what the file does—CHECK, DATA, EDIT, TOOLS, SCRN, and so forth. In general, names ending in P25 are for CAPI applications, D25 for CADE, T25 for CATI, and X25 for general functions. Within the ‘X’ files different sections of code may be compiled different for CAPI, CADE, and CATI applications based on preprocessor directives.

For the work described here we have only dealt with the screen and data handling functions (SCRNX25.PAS and DATAX25.PAS)

8. Keystroke data

Our most recent extension of Blaise 2.5 to add a function it doesn’t have in native form was to implement a keystroke capture capability. Keystroke files record every keystroke pressed by a user of an application—standard alphanumeric as well as specialized computer keystroke entries such as function keys, backspace, enter, and key-combinations like SHIFT-F4, etc.

Keystroke files are used in software usability testing generally and specifically in survey methodology studies to study what users are actually doing in the applications. Where is on-line Help invoked most often, how often does the user back up, change modes, delete answers and so forth. Usability testing of applications is an of increasing importance as studies reveal that there is often a wide gulf between what application designers think they have provided to users and what users are able to successfully make use of.

Usability testing has not been traditionally employed in survey research. One reason is that many survey designers and programmers believe that their users—telephone and field interviewers, coders, and data entry operators--will be thoroughly trained on the system and then gain substantial on-the-job experience. So, the programmers conveniently assume, why worry; interviewers will be able to master the system, even if it lacks some polish and has some clumsy elements.

Yet studies from the Human-Computer Interaction literature suggest that even experienced users fail to use key features of an application, make significant errors, and develop an adversarial attitude to poorly designed applications. (Landauer, 1995; Shneiderman, 1996). Landauer (Chapter 10) cites efficiency differences of from 10 to 200 percent between applications developed using user-centered design methods and those developed with these methods.

Another impetus for usability testing in survey research is the increasing utilization of audio computerized self-interviewing (ACASI) for data collection (O'Reilly, et al, 1994; Tourangeau and Smith, 1996; Turner et al. 1996). In this mode users who may never have used a computer before are introduced to the system and guided through tutorial application. After a few minutes they are expected to manipulate the survey application entirely on their own. Learning where these users may be having trouble based on keystroke patterns and timing can be critical to identifying where the application must be refined.

There are many different techniques for usability testing—laboratory testing, observeration, user debriefings, and others. Keystroke records provide an important additional method, one with more detailed information on some activities and, as well, one that may be less expensive and move broadly applicable than laboratory methods.

9. Implementing a keystroke capture capability in Version 2.5

For our keystroke system we wanted to be able to

- record all keys pressed by the user,
- record the timing to the hundredths of a second when the Enter key was pressed,
- identify the current question on the screen associated with the key series,

- save the keystroke series in a text file in a form that facilitated later analysis of the file.

The critical element is having access to the point at which each keystroke code is evaluated by the program. One cannot do this in Blaise III, CASES and most other Computer-Assisted Interviewing packages. Some, such as CASES, allow developers to save a 'trace' file which records every response to questions input by the user. While quite useful for testing, a trace file only save the state of the response when the Enter key is pressed. Backspacing, overwriting a mistake, and resuming cannot be detected along with other user actions.

In Blaise 2.5 adding this capability is not very difficult. In fact it is so simple that we can describe in a few sentences :

1. In DATAX25.PAS define the array ABCDE of [1..200] char.

2. In the SCRNX25.PAS there is a GETKEY procedure. In GETKEY is a while-do-end loop in which the PASCAL READKEY function is called and the result is returned in the LETTER. If LETTER is not empty a key has been pressed.

3. Add PASCAL code to save the key code to ABCDE. When the Enter key is pressed add the system time to ABCDE followed by carriage-return/line-feed codes.

4. When ABCDE approaches its maximum size, open the DOS file named KSyynndd.DAT , append ABCDE to it, close it, and ABCDE load keys from the beginning.

The actual implementation has a few additional intricacies to make the file more useful. Still, the total effort was relatively modest, probably less than 25 hours by a skilled Pascal programmer. That, in itself, should suggest that flexibility and power of this approach.

The following section displays an example of the keystroke file for a recent study.

```
{*** 11/13/96  14:30:13.38 CURRSYS:ftest1e CASE_ID:6535 MODE:MI ***}
1{4}14:30:24.64
{7}14:30:25.24
{8}14:30:25.46
1{8}14:30:26.39
{9}14:30:26.89
22{10}14:30:27.82
2 3 43[8][8]74{22}14:31:09.29
1{12}14:30:28.59
[10]{13}14:30:32.55
[[         6535]]
{*** 11/13/96  14:30:35.79 CURRSYS:ftest7f CASE_ID:6535    MODE:MI
***}
[10]{2}14:30:38.09
{2}14:30:38.70
[[         6535]]
{*** 11/13/96  14:30:51.11 CURRSYS:ftest1e CASE_ID:6534    MODE:SKIP
***}
1{4}14:30:56.22
{7}14:30:56.44
{8}14:30:56.66
1{8}14:30:57.70
1{9}14:30:58.58
22{10}14:30:59.62
{12}14:30:59.95
1{13}14:31:00.78
1{14}14:31:01.71
{20}14:31:03.25
2{21}14:31:04.24
1{26}14:31:11.65
```

One can see some of the additional refinements we included to make viewing and parsing the file more convenient. These include :

1. A header record identify the date/time the current Blaise instrument was started and information on the application's name, the case ID, and mode. Because we did not know how to determine internally from Blaise all the information we wanted to embed in the keystroke file, we added a section in the SCRNX25 code. The current directory is checked for a file named CURRSYS.TXT. If found, its first line is read and added to the header line following the CURRSYS: tag. This allows us to add-in easily any annotation information we want in the keystroke file by writing to CURRSYS.TXT before calling the application.

2. Following the header, each line contains in order the series of keystrokes entered, followed by the number of the current question field inside of curly braces {}, followed by the system time.

3. Non-printable key-presses such as function keys are displayed in readable form within square brackets. So in the following line :

   ```
   2/3/43[8][8]74{22}14:31:09.29
   ```

   the [8] means the backspace was pressed twice to change the year of birth from 43 to 74.

4. At the end of the keystoke capture for each application the KEY field is identified in double square brackets.

10. Adding keystroke capture to Blaise III

If a keystroke capture feature is of significant value to some users, shouldn't this be implemented to Blaise III? We emphatically think so. As our survey applications become more elaborate and as we venture into new areas such as audio self-interviewing and other potentially valuable new methods, the need to understand what the users are doing or failing to do become much more important. A keystroke capture capability is critical to studying that.

The Blaise III team must be the ones who implement it, adding it to the DEP data entry program. Fortunately or unfortunately, there is no back door into DEP so that a users might do it on their own, as in Blaise 2.5.

11. Conclusion

Our experience is that the unique architecture of Blaise 2.5 permits adding important extensions to the system such as the keystroke capture, and doing so is relatively simple. Of course, other extensions might not work so easily. The ability to access elementary system functions and processes in Blaise 2.5 through its Pascal source files is extremely flexible, powerful, and dangerous. For specialized needs it may provide a solution unavailable by any other means.

12. References

Landauer, Thomas K. 1995. The Trouble with Computers. Cambridge, MA : MIT Press.

O'Reilly, J. M. (1993). 'Lessons Learned Programming a Large, Complex CAPI Instrument.' A paper presented at the International Blaise Users Conference, London, England.

O'Reilly, J., M. Hubbard, J. Lessler, P. Biemer, and C. Turner (1994). 'Audio and Video Computer-Assisted Self-Interviewing: Preliminary Tests of New Technologies for Data Collection.' Journal of Official Statistics , Vol. 10, pp. 197-214.

Shneiderman, Ben. "User Interfaces for Survey Data Collection". Presentation to the International Conference on Computer Assisted Survey Methods. San Antonio TX, December 1996.

Tourangeau, Roger & Tom W. Smith. "Collecting Sensitive Information with Different Modes of Data Collection." Paper presented to the International Conference on Computer Assisted Survey Methods. San Antonio TX, December 1996.

Turner, Charles F. Barbara H. Forsyth, James O'Reilly, Phillip C. Cooley, Timothy K. Smith, Susan M. Rogers, and Heather G. Miller. "Automated Self-Interviewing in surveys: a review and research agenda," Paper

presented to the International Conference on Computer Assisted Survey Methods. San Antonio TX, December 1996.