What Was Experienced by a Skilled Blaise Programmer in Transitioning from Blaise 4 to Blaise 5

David Dybicki and Peter Sparks, Survey Research Center, Survey Research Operations, University of Michigan

This paper discusses how we transitioned to Blaise 5 from Blaise 4, taking into consideration organizational requirements and standards. To accomplish this, we had to be open to using features in the new environment and be willing to change from the prior platform.

1. New Features

Blaise 4 is essentially a single-user application that multiple users can use simultaneously but which runs as a single instance on a computer. Everything it needs to do is self-contained within the Dep.exe. Blaise 5 is well suited to work in a server environment but is complex because the functionality has been divided into separate server roles that communicate with each other (audit, CATI, data entry, resource, session, web, manager) and that may be on different servers (we typically have front-end web servers and back-end servers that handle the other roles).

Blaise 5 uses server events and listeners to communicate between the services on various servers and has also allowed the Blaise survey to raise custom server events. So, the management of Blaise surveys can be highly customized. For example, several surveys use the server events (such as the survey Completed, Expired, Aborted, and custom values) and write the resulting case status to a management database. The recipient of the events, usually a survey-specific Windows service, can parse custom events and perform complex operations, run stored procedures, write logs, and so forth that make the survey(s) operation seamless. This extension of Blaise is more than just running the survey and waiting for the program to run; it is interacting with Blaise at a new level.

Managing users within the Blaise 5 environment also changed dramatically. In Survey Research Operations' Blaise 4 CATI sample management system (SMS), users are assigned roles and groups, and passwords are managed by Maniplus using additional Blaise databases. In Blaise 5, these functions are maintained within the Blaise server manager in the Users tab. Because of how Blaise works with Active Directory, the user's password is no longer explicitly managed by SMS but can instead be synced so that it matches their login. This makes it easy for the user to no longer remember two separate passwords and places the management of users in one area.

These are just a couple of examples demonstrating significant payoffs for investing the time to learn the new development environment and its capabilities.

2. Layouts

It was not easy to assimilate all these new ways of doing the work without considering them from the Blaise 4 perspective. As a result, there was an effort to try and force Blaise 5 to look and behave in familiar ways. The reasoning was to try and make the transition smooth for interviewers and projects, but the attempt was ineffective. By analogy, it was a case of trying to put a new sporty engine and features into an older style vehicle body, and it doesn't work well. We did learn a great deal in the process, but eventually, we embraced Blaise 5 and took advantage of the new capabilities to expand our capabilities. Layouts in Blaise 4 can also be complex (mode library, menu, configuration, datamodel properties), but not to the same degree as in Blaise 5. The Blaise 5 Resource Database, .blrd, contains all of the Blaise 4 display items but is organized in a way that requires a different way of thinking. Using the layout view and property windows in the Control Centre was key to working with the layouts. Initially, we struggled

to relate all the different pieces together (templates, properties and parameters, layout sets, master pages, expressions, field references, etc.). However, as we gained experience, these tasks became more manageable. We learned that “less is more” (use less customization per page and more standards).
We realized from our experience in using Blaise 4.8 that we needed standards in the new environment. It was essential to hash out what is required, and similar to having a universal starting mode library file, we worked on (and are still working on) a standard blrd file. We have found that we “borrow” features that work from one study to another but maintain standards for the look and feel of the survey where possible. This builds solid and predictable input screens familiar to all interviewers, study staff, and programmers.

3. Texts and Interfaces

Working with screen text and layouts in Blaise 4 is well known and comfortable. Managing how a particular question looked on the screen meant working with the mode library editor (InfoPane, FormPane, and Grids), font enhancements (like “@B” for bold and “@I” for italic text), datamodel properties (language character encoding and culture), and the menu editor (menu items, panels, and actions). Because of our expertise, surveys can be programmed quickly and efficiently.

But Blaise 5 opened a new world to managing the survey look, feel, and user interaction. The Resource Database, along with the Layout tab and properties panel stored in the .blax.layout file and the Layout section in the source code gave a great deal more flexibility. Font definitions are now named, language text can be copied and pasted directly into the source code (no ANSII transformations needed), and HTML-like commands can be used for additional formatting with text areas.

There are also runtime interactions between the survey and the interface via field reference mappings, languages, roles, type references, function declarations, and template parameters. The complexity of putting text on the screen has increased, but the reward is having an adaptive context-adjusted display that couldn’t be done in Blaise 4.

4. Blaise 5 Resource Database

The Resource Database can be considered its own programming area of expertise. Many new concepts are presented: layout sets, templates (such as master, field pane, input controls, and grouping), template parts, default templates, style elements, styles, text roles (embedded), media, master pages, containers (such as grids, panels, groupbox, and areas), controls (like labels, buttons, image, and hyperlink), size allotment (stretch, auto, percent, and pixels) and embedded sizes, applicability conditions, parameters, events, actions, shortcuts, and custom properties.

All of this presented new challenges and forced us to think in novel ways about defining the look and feel of our surveys. Although some parts were easy to understand (styles, style elements, and font definitions), others were harder: text roles, field references, texts, media, templates, and resource sets. One of the most significant concepts we had to grasp was the use of containers within a template and how the properties of auto, stretch, undefined, *, %, and pixels all worked, especially when a container was within a container with different size attributes. It took many hours of trial and error before we understood how everything worked.

5. Abandon Prior Tools and Features?!

Many tools and features are tailored to the environment being used. For example, in Blaise 4, there’s the Menu Editor, Data Centre, Bascula, Cameleon, Delta, X-Tool, Mode Library Editor, Dep Configuration, Hospital, and Database management (copy, delete, move, rename, create). In Blaise 5, many of these are no longer needed because the environment is very different. For example, all Blaise 4 menu functions (assigning layouts to fields, layouts of pages, menu items, actions, and panels) are instead found in the Blaise 5 Resource Database, which is managed in the Layout view of the .blax file, or are synchronized back to the source code in the Layout section. As a result, there’s no “Menu Editor” in Blaise 5, and so it is strictly a Blaise 4 tool that’s no longer “needed.”

Other utilities, such as Cameleon, essentially became redundant when full access to the meta information in Manipula became available. Blaise 5 moved all of its data storage to relational databases, so data management features (like Hospital) were no longer needed or were moved to new tools. For example, the ability to monitor databases is in Blaise 4 but is accomplished somewhat by the Session Viewer or by running queries against the audit or session data.

Some features of the Blaise 4 language, such as the alien procedure, were not brought forward to Blaise 5. This is used to make custom entry boxes, scan through the datamodel structure for additional checks, and perform logging or other special actions but is no longer needed because the Resource Database has conditional logic and access methods that provide the same functionality.

While knowing how to use tools and features for a particular environment is always beneficial, allowing them to be left behind when moving to a new one is necessary.

6. Programming Standards

At the Survey Research Center at the University of Michigan, we have had many years of programming in the Blaise 4 environment and have developed a set of survey programming specifications and standards. These are used as a guide for successful software development and are applied to every Blaise survey we do. The longer we used Blaise 4, the more complete our standards became.

The challenge in transitioning to Blaise 5 was to make similar standards for code writing and presentation to the interviewer/user of the instrument. This task is complicated to accomplish when the capabilities are unknown and the programming environment is new. We used the Blaise 4 standards as a starting point, but since then, we have been adapting and changing them to fit the new system. These standards are critical for helping programmers build an instrument that meets and exceeds the client's expectations.

We discovered that we had to learn the new system comfortably before effectively writing new standards. We started with what we knew and then adapted them as needed.

7. Programming Concepts Changed

Although it’s very comfortable to remain in the old way of thinking and using programming methods and utilities, it is best to change with the new environment. However, there’s a price to be paid, and that price is a steep learning curve.

For example, when introduced to the new GROUP structure, it was confusing, and we assumed it functioned the same way as BLOCK but with some tie-in with the layout. Instead, the GROUP structure is a hybrid where it allows fields to be displayed together using a grouping template, but in terms of implementation, the RULES for the fields involved belong to the GROUP (like it is within BLOCKs). But it is unlike a BLOCK in that fields and auxfields *are not* stored within the GROUP level but are stored at the same level where the GROUP is defined. This “old” way of thinking caused a great deal of head-scratching until the new concept was learned and our assumptions were abandoned.

In Blaise 5, there were many new added features that alter how the surveys are programmed: constants, user-defined attributes, field properties (e.g., isVisited), conditional logic within templates, field parameters, role texts, and many more that control the logic flow and interface. The key to programming is knowing exactly how these new features work and using them correctly (not the old ones).

As we became better experienced and more confident in the Blaise 5 environment, we took advantage of new features and methods to improve the surveys and performance. We let go of old strategies that no longer worked well or were prohibitive to maintain. In addition, we continued to learn by trying new features and methods. We worked on improving Blaise by reporting bugs and asking for help when we couldn't figure it out independently.

8. Interview Interface Changed

In Blaise 4, we were comfortable with a display of question text at the top panel (InfoPane), information about the question answers in the middle panel (code responses or range information), and then entry for the field in the bottom panel (FormPane). There was just one question per page, and navigation between fields was all done in the field pane. DK/RF/Remarks symbols appeared next to the entry box, and help popped up in a separate window.

In Blaise 5, all of this changed: we could freely rearrange all these parts, the entry field could be placed practically anywhere, there could be multiple entry fields, help appeared in a different text box, controls could be hidden/shown on demand, menu controls were largely removed, previous/next buttons became the norm, and screen displays became dynamic.

This impact is a far more interactive display for the interviewer and the respondent and matches what is used in modern applications. It's important to "keep with the times," even though the basic functionality is present in both Blaise 4 and Blaise 5. That means being a Blaise 5 programmer also means another set of skills is needed in the programming toolbox, and that is being a web designer.

9. APIs and Versions

Blaise 4 is very rich in terms of providing the ability to dive into the metainformation of the datamodel and retrieve everything: every bit of rules logic, parameters, fields and attributes, texts, languages, layouts, code texts, and so forth. Blaise 5 is also very rich and provides all the same information.

However, trying to navigate through the datamodel using the application programming interfaces (APIs) is very different between the two platforms, and it simply did not work to apply Blaise 4 API methods to the Blaise 5 API. We found the best method to learn the new API was to step through sample programs using the .Net debugger and examine the areas of interest (objects, properties, and methods). Once we found how to retrieve the information and the context in which it was used, we implemented it within our own utilities and built up our expertise.

We also had to learn what files and DLLs had to be present for running our utilities and licensing requirements to deploy and use them. Part of the lessons we learned is that any program that uses a version of Blaise tends to be version specific. That is, suppose a data utility is compiled using Blaise 5.6, and then is used against a Blaise 5.13 database; it may not function properly, but a compiled utility using 5.13 may work with Blaise 5.6 data. Blaise extends the Blaise 5 API with each new release so utilities can reference older methods using the new API.

We recommend upgrading utilities to use the new API (as appropriate) and methods whenever possible to take advantage of bug fixes and new features.

10. Data Storage Expanded

In Blaise 4, there was one "flavor" of data storage: a proprietary format that consisted of several files, along with additional utilities to manage the storage (such as Hospital, Control Centre copy, move, rename, and delete). All this changed to a relational database in Blaise 5. In addition, many more options were given regarding internal table structures with data partitioning (Stream, Flat No Blocks, Flat Blocks, In-Depth, and single table), storage types (Non-generic and generic), and platforms (SQLite, SQL Server, Oracle, Excel, MS SQL, among others).

In addition, the audit information changed from a file into a database with two tables. Both require parsing, but working with the database using SQL queries is much easier.

What constitutes a data change is also different between the two versions. Changing the name of a field in Blaise 4 is not considered a data change, but moving that field to a different position or changing an attribute will require a data migration. In Blaise 5, the opposite behavior occurs (name changes require migration, but moving names/changing attributes don't).

Because of the relational database, Blaise programmers had to pick up additional database skills and understand that the data was no longer directly accessed but instead went through a data link file (connection properties) managed by Blaise. Hence, if the data link referenced a survey on a server, no matter where the .BDIX resided, the data could (potentially) be retrieved, modified, and written. As a result, we learned a great deal about database user accounts, logins, schemas, and the like.

An area that we have not explored yet but are willing to try is cloud storage and hosting. It is possible for specific surveys, but additional areas must be scrutinized: security, availability, reliability, cost, etc.

The basic concepts of working with data remain a great starting point, but the new environment requires specific knowledge of how the system works. Only the minimum of data manipulations can be accomplished without the additional training.

11. Audit Information

Likewise, the audit information in Blaise 4 was no longer stored in a single text file (or even multiple files or multiple cases within one text file) but instead is stored in a relational database in Blaise 5. Our utilities that parsed the keystroke-level information started as the base for new software but quickly changed and became customized to the new environment. However, the function of our utilities remained the same: to analyze the data for section and field timings, recover cases, and understand areas where a given survey might be having issues.

12. Session Data

In Blaise 4, just one database contained the initial preload, working cases, and completed cases. If a case needed to be restarted, it was simple to run a Manipula Script. However, the Session Database (the concept first introduced in Blaise IS) took more effort to work with. The data from the main database is copied to the Session Database, and then all further breakoffs and resumes use the Session Database. We quickly discovered that we must retrieve cases from both Main and Session to get a complete snapshot of production.

This has several important implications: any data pulled from the main database will not have partial case data unless the session data to the main database is explicitly copied (usually by a Manipula Script), or the case has been completed. Therefore, tracking the overall status of a web survey can be inaccurate when only looking at the main database. In addition, the Session Database stores the "state" of the survey: every local, auxfield, and field property is present in the data so that when a case resumes, it is in precisely the same place where it left off. This caused all sorts of problems for our Blaise 4 way of thinking (auxfields and locals were initialized every time a case started) until we understood that we had to explicitly manage this "reset" by passing in a parameter/flag to initiate code to perform this function.

The automatic storing of case data in the Session Database has proved to be a bonus for some situations and a hindrance for others. For example, when case data is accidentally reloaded to the main database, it is possible to restore cases quickly from the Session Database by copying them to the main database. However, changing preloaded incentive amounts is more challenging because the session data cannot be written directly. (There are a few methods to address this: one way is to download the session data to the main, delete the session data, and then update the main data with the new incentive. When the case is restarted, a new session will be created from the main data.)

Like other new methods in Blaise 5, this initially caused us problems from our lack of understanding, and we tried all sorts of ways to get around what we thought was a restriction. After we gained the necessary knowledge and worked with the feature, then we gained the benefits it offered.

13. Manipula

Manipula has been the mainstay workhorse for our use in Blaise 4. It has been critical for our processes: loading preload, moving and updating data, exports, alien procedures, specialized parsing, interaction with CATI, creating crosswalks from the meta information, and so much more. We literally could not run our surveys without this valuable tool.

Blaise 5 Manipula is just as valuable. However, like many areas in Blaise 5, it was the same and not the same. During the transition period, as Blaise 5 Manipula was still being developed, we had to learn/relearn the language because it was a work in progress. Language features were removed because those areas were no longer part of the Blaise 5 system, and others were added to work with new features (like write interceptors, CMA, and session data). Maniplua was also updated to use Action Setup Manipula Scripts or Manipula Dialogs.

Because Blaise 5 Manipula is so diverse, powerful, and flexible, we are still learning its capabilities and are hard at work to become even better experts. For example, many API functions are available within Manipula, but it takes advanced programming skills to dive into them.

14. Blaise Control Centre Changed

A big challenge has been for programmers to learn the Blaise 5 Control Centre, and it seems that everything changed. We had been familiar with menus, file types, projects, tabs, and where all the parts could be found. However, in Blaise 5, there were now solutions, packages, different previews for different environments, tool and property windows that can be dragged/dropped/resized, different file types, context-sensitive menus, and so much more. It took some time to become familiar with the panel arrangements, especially the placement of the Source, Meta, Settings, and Layout tabs for the .blax file.

Editing in the source code was different; some familiar keystrokes were no longer present, new ones were added for new features, and a few changed how they behaved. This is typical for different software environments.

Regarding functionality, Blaise 4 and Blaise 5 perform admirably for what they're designed to do. Multiple parts of Blaise 4 are integrated into Blaise 5 in a way that makes sense. For example, the Delta tool in Blaise 4 shows diagrammed program flow for a datamodel. It works with the datamodel and acts like a "plug-in" added to the Blaise system. In Blaise 5, it's integrated into the menu by opening the .blax in the Control Centre—selecting the Meta view—then clicking the Statement graph button.

There's additional code handling functionality that's built into Blaise 5. Some handy shortcuts are comment/uncomment code, bookmarks (also found in Blaise 4), and source structure browser. In Blaise 5, a new editor concept of code snippets can be customized to save time writing standard programming code. The Find/Search & Replace also have been changed and function differently. For example, there's no "enhanced" find/replace in Blaise 5 because it's controlled by the scope of the action (selection, document, project, and solution).

Autocompletion and parsing while editing are features that are strictly in Blaise 5. Because of the new editor and functionality of the Blaise 5 Control Centre, there has to be a period of adjustment before becoming proficient. The transition period may be frustrating and slow, but the result is a greater ability to program efficiently.

15. Interviewer Help

The help file for Blaise 4 has always been a mainstay for our interviewers and desired by our project staff, but it has become problematic: support for the help file format has been deprecated. This means there is extra setup on the interviewers' laptops to install software/DLLs, and sometimes extra effort for the Blaise programmers to keep this feature working. It may become unusable if the operating system no longer supports the software to run the help.

In Blaise 5, the help moved into the survey itself as a role text and can be mode specific and easily translated in multiple languages. The change means the staff no longer edits an RTF file, but the programmer has to place the text into the survey. The interviewer views help differently by typically using a button on the screen to show the help text on the page, rather than seeing the help in a separate window.

16. All Interviewing Modes Possible

Blaise 4 had one mode with data collected in different environments (laptops, desktops, and via management systems), so each survey was typically programmed for each particular survey mode. However, Blaise 5 has true modes (CAPI, CASI, CATI, etc.) that can be used in different environments, such as iwer-assisted CAPI + DEP on remote laptops, iwer-assisted CAPI + Web on local computers, respondent self-collected CASI + Web on desktops/phones, and so on. Typically, the attributes in our Blaise 4 environment have been defaults of DK, RF, and NOEMPTY, but Blaise 5 is richer in that all these are contained within one datamodel and the mode can be set at runtime. This gives excellent flexibility and consistency for the surveys.

17. Mode Switches

Because our Blaise 4 data was collected in essentially one mode, switches between modes was rarely an issue. However, planning for mode switches has become a necessity when using Blaise 5.

As noted above, Blaise 5 allows multiple modes in one survey, and the attributes of self and interviewer assisted tend to be opposite of each other. Going from DK and RF allowed to not allowed implies these values are dropped, while going from EMPTY to NOEMPTY means questions that have been skipped will now be asked. Different methods have been used to implement mode switches, such as blocking completed sections, restarting partially completed sections, copying unique mode-related fields between mode switches (and assigning correct values), and not allowing mode switches.

18. Security Enhancements

The Blaise 4 data is stored in a proprietary format that is readable via the data viewer, Manipula, and BCP programming—all different ways that access is granted openly. Security on the data is maintained at the file and directory levels.

However, in Blaise 5, the data may still be in a file (.bdbx using SQLite—and the same file security methods apply) or in an SQL server or other relational database software where it is referenced from a data link file (.bdix). Learning how best to secure the survey data (audit, session, and main survey data) in the new environment was essential. This .bdix can be put anywhere and the data can be accessed (with caveats) as long as there's a connection to the server hosting the data. Therefore, careful attention to the location of production .bdix files is needed to avoid unauthorized access to respondent data; that is, production .bdix files should be kept in secure locations. Blaise 5 allows password protecting the .bdix file to limit this possibility, which is a feature not available in Blaise 4. Additional welcome security measures have been implemented in general to the Blaise system, such as in the AD synchronization, server manager, and server manager roles and user passwords.

19. Increased Use of Client/Server

Blaise 4 (using Blaise 5 terminology) is a thick client survey (survey + data collection is done without server contact). In Blaise 5, most surveys use a thin client (web survey with heavy server use) to retrieve survey pages and store survey data; otherwise, the Blaise 5 thick client is similar to the Blaise 4 survey. We did not grasp this concept initially, and only after working with Blaise 5 after a while did it make sense. We had to move out of the prior thinking and into the client/server world. We're now much more familiar with the capabilities of Blaise 5 and have been able to use it for self- and iwer-assisted surveys in DEP, custom Dep, and client/server. It took some time to gain expertise and understand how the pieces work. We had to invest time and effort into building login applications (portals) for respondents, learning IIS management, Blaise server management, building a custom DEP, and writing upload and download interceptors in Manipula.

20. Conclusion

Blaise 4 and Blaise 5 are rich, complex, versatile, adaptable, and robust, and they can handle difficult situations. In moving to Blaise 5 we viewed nearly all the new features, methods, editors, and utilities by looking through the Blaise 4 lens, which, in effect, distorted the new reality, caused more work, decreased our productivity, and limited what could be accomplished. As we gained understanding and embraced the new ways of working with the system, we enjoyed more excellent capabilities in our surveys, faster programming, and more possibilities. Not everything is peaches and cream in the new environment, and there are still difficulties to overcome, but we have grown to appreciate Blaise 5 without the Blaise 4 overtones. We can apply these lessons to any new software or environment and admit that we still have much more to learn.