

# Blaise 5 Improvements for Large CAPI Surveys

*Peter Stegehuis, Siu Wan, Naxin Zheng and Ryan Webb, Westat*

## 1. Introduction

For some of the largest and most complicated Blaise CAPI surveys, changing from Blaise 4.8 to Blaise 5 is a long process. As we all know, Blaise 4 and Blaise 5 are different enough that, at least for complicated surveys, there is no easy and quick path from one to the next version. There are many facets that play a role in determining if, when and how to implement such a momentous upgrade. In this paper we will focus on the Blaise programming side.

From that programming perspective, the most important issues are ensuring that specific features can be made to work, and at the same time that overall performance is satisfactory. After all, Blaise 4.8 may be limited in what it can do compared to Blaise 5, but it is fast in what it does. A good overall performance is important to keep interviews flowing, reduce respondent burden, and thereby keep response rates as high as possible.

This paper delves into some of the testing we did for CAPI performance, and how Blaise 5.14 and now version 5.15 address those issues.

## 2. The Process of Updating a Long-Running CAPI Instrument

When we are discussing complex CAPI instruments, we are automatically talking about big projects, and with that multiple stakeholders. Besides feasibility of the update itself – can the new software handle the complexity in a satisfactory way – there are many parties and considerations that play a role in whether and when to update the CAPI instrument.

There are obviously communications with the client (internal for a national statistical agency, external for a contractor), first and foremost about the risks regarding continuous operations and data quality for the ongoing survey. Subsequently, there is a big investment involved, in time and money, for many aspects, including:

- A possible redesign of screens and sections
- Programming architecture
- Programming and testing
- Creation of interviewer training materials
- Interviewer training

Assuming there is a budget and a schedule to go ahead with the conversion from Blaise 4.8 to Blaise 5, there are different approaches possible for the programming and testing, to end up with a complete Blaise 5 CAPI instrument.

One approach is to just have different groups program a number of interview sections, see how things go along the way and hope that things work out in the end. However, that leaves the risk of not being able to satisfactorily solve a specific issue, or of finding that you have an overall performance issue, at the tail end of a long and costly process.

A much more proactive way to mitigate risks, in our view, is to start by creating small prototypes, each one mimicking an important and complex part of the instrument.

This approach is comparatively easy for some aspects, and much harder for others. For instance, if at some point the CAPI instrument needs to shell out to a separate executable, you can fairly easily create a prototype that does just that. The outcome is positive if you can shell out and get control back to CAPI, if the whole process is smooth and fast enough, and if the CAPI instrument can pick up the results of what was done in the separate process successfully. We created a number of prototypes that each tested a specific feature that is important to us, and found that Blaise 5 (version 5.13 at that time) handled them well.

The much harder element to test in a prototype at the outset is one that can at least fairly accurately test the overall performance of a big and complex CAPI instrument. After all, you can't test the CAPI instrument with all its sections and complexities until you have programmed them and put everything together. Instead, to test performance early on in the programming process, you'll have to rely on prototypes that can be expected to mimic at least some aspects of real life complexities.

In the remainder of this paper we will focus on two prototypes that tried to test two performance-related metrics:

- The time it takes to go from one screen to the next, comparing a small CAPI instrument to a very large one (small and large in terms of the size of the metadata)
- The time it takes to go from one item on a screen to another, comparing screens with few items on a page to screens with very many items - especially grids with potentially many rows

We will discuss the 'prototype approach' for each, the issues that we found, and the solutions that came in subsequent Blaise 5 versions.

### **3. A prototype for sizeable datamodels**

In Blaise 5.14, Statistics Netherlands implemented the smart session data update to improve runtime performance of large data models. Now that we have 5.15, we did some tests to measure 5.15 as well.

We created 5 data models for comparison:

1. Model 1F with a person array of 25 and an event array of 500
2. Model 2F with a person array of 50 and an event array of 25000
3. Model 2N with an event array of 500 nested in a person array of 50
4. Model 3F with a person array of 50 and an event array of 50000
5. Model 3N with an event array of 1000 nested in a person array of 50

In other words, models 2F and 2N are the same size, but the arrays in 2N are nested while the arrays in 2F are side by side. The same is true for models 3F and 3N. In terms of size, model 1 is the smallest, models 2 are bigger, and models 3 are the biggest. The person array has 28 questions displayed in 7 pages; the event array has 11 questions displayed in 3 pages.

What we want to test with these different datamodels is how long it takes to go from one screen to the next, with these two screens being exactly the same ones across the different datamodels. That way we think we can get a good understanding of the role that the size of the metadata plays, and we can form an opinion on how well Blaise 5 handles very large CAPI instruments.

We then worked a couple cases with preloaded data and a couple cases without preloaded data, with each data model respectively. We used the audit trail data from these worked cases to calculate field/page loading time.

Given a data model size, we found the differences between with preloaded data and without preloaded data are not significant, the differences between nested arrays and side by side arrays are not significant either. Also, in pages with multiple questions, the time to load subsequent fields within the page is also so trivial (less than a thousandth of a second) that it's hard to make meaningful comparison. As a result, the discussion here will focus on page loading time given Blaise version and data model size.

As you can see in Figure 1, while we can say 5.15 did better than 5.13, we cannot really say it did better than 5.14. However, we also cannot really claim this as a statistically significant conclusion since our sample size was so small.

**Figure 1. Average page loading time in seconds**

Data model size	Average page loading time			Loading time comparison		
	5.13	5.14	5.15	5.15 : 5.14	5.15 : 5.13	5.14 : 5.13
1 (25 + 500)	0.162	0.265	0.149	0.562	0.919	1.636
2 (50 + 25K)	2.086	0.382	0.419	1.097	0.201	0.183
3 (50 x 500)	2.721	0.459	0.749	1.631	0.275	0.169

In addition to page-to-page performance, we also noticed that, for large data models, it could take more than just a split second for Blaise to show the first page of the instrument and have the first question ready for answer. So we also measured the time it took for the first page to show, i.e. from the moment the DEP.exe is executed to the time the first question is ready for data entry. Figure 2 shows that 5.15 in general has favorable performance on loading the case up.

**Figure 2. Average first page loading time in seconds**

Data model size	Average page loading time			Loading time comparison		
	5.13	5.14	5.15	5.15 : 5.14	5.15 : 5.13	5.14 : 5.13
1 (25 + 500)	3.054	7.995	1.956	0.245	0.640	2.618
2 (50 + 25K)	10.389	12.273	6.733	0.549	0.648	1.181
3 (50 x 500)	11.707	16.412	11.525	0.702	0.985	1.402

An additional note is that, to run the 5.15 cases, we had the setting of DefaultProtocol in the file Dep.appsettings.json changed to "Rest" so that we could use the Dep.exe with REST. We have not had a chance to compare the performance between protocols WCF and REST in 5.15 so far.

#### 4. A prototype for grids with many rows

While tables in Blaise 5 look a lot more sophisticated than in Blaise 4, they also took a lot longer to load when we first tried to put some of our household roster tables in Blaise 5.14. After we expressed our concerns to Statistics Netherlands, they promised big improvements in Blaise 5.15. We believe that they delivered.

To see how things improved, we created a CAPI instrument with only 5 pages: 1<sup>st</sup> page to ask for size of household that determines the number of rows in the tables; 2<sup>nd</sup> page to confirm the list of household members; 3<sup>rd</sup> page to a relatively wide table to collect name, sex, date of birth, etc; 4<sup>th</sup> page to a wider and even more complicated table with routing to collect marital status, race and ethnicity, languages spoken, etc; 5<sup>th</sup> page to confirm and end the interview.

After we completed 5 cases each with 1, 5, 10, 15, 20, and 25 table rows respectively, we calculated the loading time per field with the audit trail data. The loading time per field is determined by the time it took from a LeaveFieldEvent to the very next EnterFieldEvent.

Figure 1 shows significant improvement in Blaise 5.15.

**Figure 1. Overall performance**

# of table rows	Average loading time per field in seconds		5.15 : 5.14
	Blaise 5.14	Blaise 5.15	
1	0.404	0.232	0.572
5	0.404	0.251	0.622
10	0.543	0.390	0.719
15	0.709	0.485	0.684
20	0.860	0.609	0.708
25	0.960	0.745	0.776

Although improvement was obvious in general, we also noticed that it takes significantly longer to load a page than a field. As we can see in Figure 2, given the same Blaise version and table size, the field loading time is insignificant compared to the page loading time.

**Figure 2. Page vs. field loading time**

Blaise Version	# of table rows	Average field loading time in seconds	Average page loading time in seconds
5.14	1	0.004	0.955
5.15	1	0.003	0.486
5.14	5	0.006	1.207
5.15	5	0.002	0.642
5.14	10	0.007	1.545
5.15	10	0.002	0.961
5.14	15	0.006	1.822
5.15	15	0.002	1.287
5.14	20	0.008	2.209
5.15	20	0.002	1.611
5.14	25	0.008	2.476
5.15	25	0.002	1.915

We then looked at the page loading time further for only the pages with tables. The page loading time is determined by the time it took to an UpdatePageEvent from the very last LeaveFieldEvent. Since a page can be reload after a critical field in a table is answered, the UpdatePageEvent can happen many times per page if critical fields are used in some columns of the table.

Figure 3 shows that Blaise 5.15 loads pages with tables significantly faster than Blaise 5.14. However, there also seems to be a pattern that Blaise 5.15's edge gradually diminishes as the number of rows of the table grows.

**Figure 3. Table page loading time**

# of table rows	Average loading time per page		5.15 : 5.14
	Blaise 5.14	Blaise 5.15	
1	1.049	0.476	0.453
5	1.241	0.650	0.523
10	1.568	0.966	0.616
15	1.843	1.295	0.703
20	2.230	1.619	0.726
25	2.499	1.925	0.770

While we do not know whether there would be a certain point where Blaise 5.15 cannot outperform Blaise 5.14 in loading table pages, a table with too many rows would not be user-friendly and should probably be avoided.

## **5. Conclusions**

Creating the prototypes was the preferred way for us to get a good idea of how Blaise 5 would handle an extremely big and complex CAPI instrument. This was true both in terms of testing very specific elements of the CAPI questionnaire, as well as for the performance tests for pages with many elements and for a questionnaire with very large metadata.

Issues we found when first testing with Blaise 5.13 were shared with the Blaise team at Statistics Netherlands. As we have maybe come to expect from them, these issues were subsequently dealt with very effectively in versions 5.14 and 5.15.

We think these improvements will benefit all Blaise users, especially as the Blaise team intentionally made these changes for all modes, not just for CAPI. We very much appreciate their efforts, and feel much more at ease about the capacity of Blaise 5 in handling the largest of our CAPI surveys.