Blaise Questionnaire Text Editor (Qtxt)

Grayson Mitchell, Statistics New Zealand

1. Abstract

Qtxt is a program designed to reduce the amount of work involved with the production of large questionnaires. **Qtxt** achieves this by enabling the questionnaire designers to edit question text without the need to know about Blaise syntax. This has reduced the amount of communication required between the questionnaire designers and the application developers.

Another key feature of **Qtxt** is its ability to handle multiple languages (including Chinese). The **Qtxt** program always displays English (or a selected base language) as a reference, and the new language is entered in another pane. The aim was to provide external translation agencies with a system to translate question text alternative languages.

Qtxt also allows the user to edit question fill text and we have fill text to be set up as category types. This means that all question texts are all stored in one place, and more importantly are easily editable with **Qtxt**. Another benefit of this approach is that our fills can easily be written in multiple languages without needing to maintain the Blaise code for the variables involved.

The question texts are structured. Each question belongs to a module, and each module belongs to a questionnaire (which we store in Microsoft Access format – though they could easily be stored in other formats). After exiting **Qtxt** the files are exported into a Blaise source file which can be compiled.

Qtxt allows the user to edit question text in an easy to use environment, similar to many other rich text editors.

The evolution of the Blaise API has allowed for more possibilities in the future (for instance the ability to read the ***modlib*** using the API). Statistics New Zealand is looking for opportunities to improve our questionnaire design process and **Qtxt** has helped with this.

Success Criteria

- Reduced involvement of technical staff in text changes
- Ease for non-programmers to change Blaise text
- Ease for external agencies to change Blaise text (such as translators)
- Allow non-programmers to change formats/ layout of text
- Integration into existing tools (e.g. our Survey Management system)
- Management of text changes on stand alone PC's (e.g. distribution to translators)
- Easy to use system for code fills in text
- Export text to Blaise code
- Edit multiple languages (including Chinese)
- A stand alone product

2. How Blaise is used in SNZ

Statistics New Zealand (SNZ) use Blaise mainly for Social surveys. These fall into two categories: paper questionnaires and electronic questionnaires. Most of the paper questionnaires have been around for a while, and the Blaise data entry systems were also developed some time ago. In recent times we have been writing much more complicated electronic questionnaires. With these electronic questionnaires we have incorporated new technologies such as the Blaise API. Blaise tools such as Chameleon and Manipula are mostly used with the older systems, or the odd ad-hoc job.

The Questionnaire Design Consistency (QDC) are given 'topic specs' that detail the outputs expected from the questionnaire. They then use these specs to devise their questions. If the questionnaire is an electronic one then this design will always be in the form of a flow diagram. Once they are happy with a module (or section) then they deliver the flow chart to the programmers (Technical Application Services) who write the code. The module is set up so it can be run in isolation. Then QDC test the module. When they find errors they tell the programmers about it, and then the programmers go and fix it.

So in Statistics New Zealand our questionnaire designers do not program Blaise, TAS do all of the programming. One of the disadvantages of this approach is that TAS can spend a lot of time on textual changes, which is a waste of resources. Training our questionnaire designers how to program Blaise might resolve some of those issues, but would introduce a whole lot more problems; managing the different tiers of Blaise programmers for one.

3. Brief History of Qtxt

So as you can see a lot of time is spent in communicating small changes to the programmer. The programmer tends to get frustrated at they get a stack of changes, and one page might be specifying that a full stop is required. Then from the QDC persons viewpoint they get annoyed because it takes a week before they get to see the program with that full stop in place.

This is the situation that made me come up with Qtxt. I did not want to make any question text changes, but I did not want QDC staff to be mucking about with our code at the same time. So I invented a tool that enabled QDC to edit text without having to worry about Blaise formatting or code.

Qtxt version 1 was a Visual Basic program. It stored the question data in individual Blaise files, and translated the formatting code into rich text for editing and then back again on saving the document. Then there came a need for us to do a multi-lingual questionnaire. Qtxt proved not to be robust enough to handle all of the languages, and the text manipulation became unwieldy. The scope of the project had grown considerably since its conception.

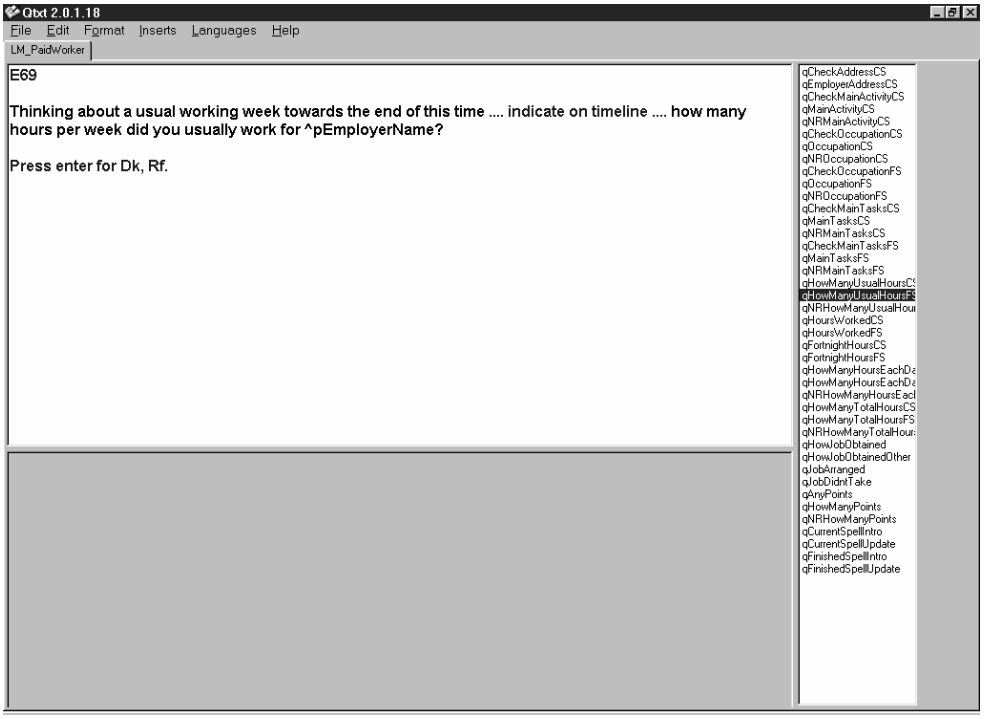
Qtxt version 2.0 is our current version. It is written in Delphi, we changed to Delphi after discovering several shortcomings of Visual Basic 6. It stores all of a questionnaire's text into a database as rich text, and so is far more robust. It allows the editing of fills in multiple languages and can handle Chinese and other languages. In fact a lot of its design and functionality have been put in place for the express purpose to make multi-lingual questionnaires easier to produce.

4. Qtxt-User interface

Qtxt main function is a text editor, and inherits a lot of its functionality from the 'TRichEdit' class in Delphi. There have been a quite a few enhancements made to make question text editing more user friendly. On the picture below you will see that on the right hand side of the screen there is a list of question names. This list enables the user to select what question they want to edit, and the text for that question is loaded into the text box.

This means that there is no possibility of the wrong text being edited. Also you will note that no Blaise code is visible. There are no @B's and no type definitions, this makes Qtxt very intuitive to use, and little or no training is required for general text changes. Your average user will be only use the navigating features to move through the questions, typing amendments, or changing the text colours. As you navigate through the questions changes are automatically saved to the questionnaires database, and when you close the Qtxt program a Blaise formated file is exported to the appropriate directory.

There are other standard type functions too, like you can add/delete questions and even modules. But these are usually reserved for the programmer rather than the standard user.



5. How does Qtxt interact with Blaise?

The text itself is stored, as rich text in tables, so is not accessible to Blaise. To keep the Blaise code always up to date Qtxt always exports a copy of the question text in Blaise format after it has been edited in Qtxt. This means there are essentially two copies of the same text, the master copy in the Qtxt database, and the local Blaise copy.

To keep this export system simple we have separated the question text from the actual code. This means that each module has an extra include file with a .Qtxt extension. This has been adopted as a SNZ standard. E.g. a .Qtxt file:

```
Fields
   qCheckChild
      "@/·
      @/I·have·your·name·as·the·person·who·can·answer·a·few·questions·about·^aW
holeName.·
      @/·@B·
      @/Press·enter·to·continue@B."
      : String,EMPTY

   qCheckName
      "@/·
      @/Can·I·just·check.··Is·that·child's·full·name·^aWholeName?”
       : tNRYN
```

Note: When we exported to Blaise we chose to export the <ctrl .> character rather than a space. This proved to be a real problem when we came to use Chinese with Qtxt. Chinese (and other Unicode languages) require that key for a character, so I had to make the exporting of <ctrl .> a option. The reason why we wanted to have the dots is because sometimes Blaise removes spaces when the code is run, an example being the that two spaces after a full stop becomes one. The users did not like that.

6. Advanced Features - Fills

One thing we have found to get out of hand in the past is text fills. With the standard way they are programmed it is not simple for a non-Blaise user to edit the values of fills. So what I came up with is a way to write fills in the Blaise code that does not include the actual text to be displayed. This has several attractive benefits. First we can manage and see the fill values much more easily as they are in one place. And we can also have multilingual fill values. To do this I set up a fill include file that lists all of the fills, e.g.

Type

```
iWasIs =
  (I1        "was",
             I2        "is")
```

Then in your code it is just a simple matter of defining a variable, and assigning to either I1, or I2. You don't deal with any text at all within your code. For users to edit the inserts can use Qtxt in much the same way, there is just an extra listbox with the fill definition. Also when encountering an insert in some question text they user can press <ctrl-s> and Qtxt will take them to that insert to edit.

7. Advanced features - Multilingual

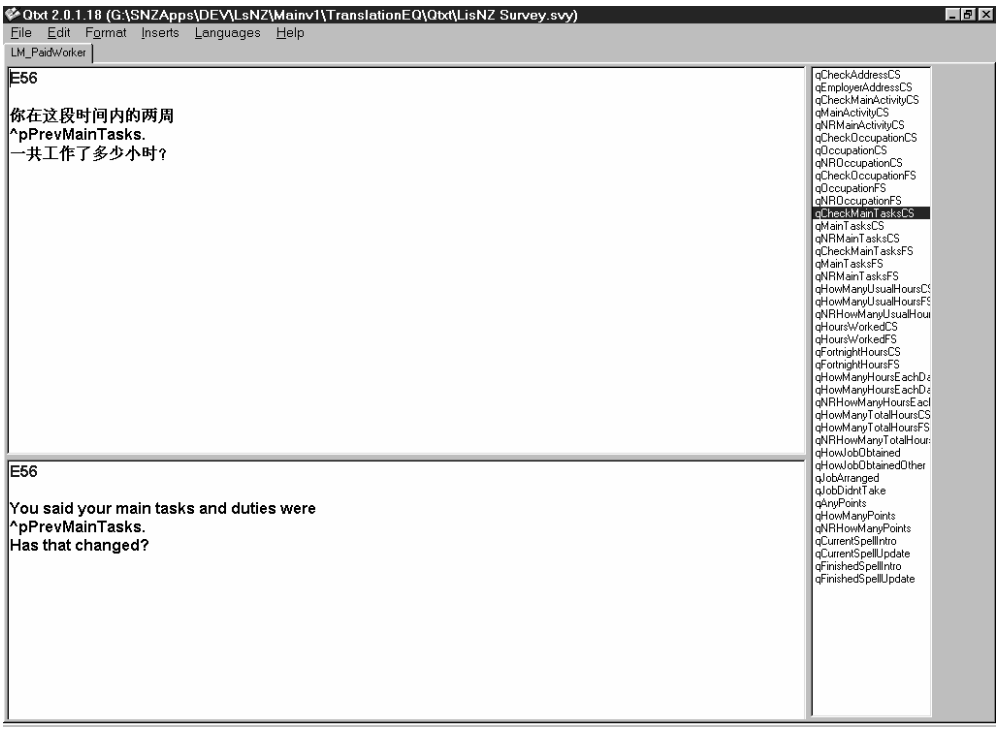
Qtxt has been designed with multi-lingual questionnaires in mind. When editing an additional language the bottom of the screen shows the original language test to aid in translations as you can see below .

Logistically translating questionnaires externally can be a nightmare. I will not go into the problems we have gone through here, but there have been many. Foreseeing technical issues is extremely difficult when most of the developers only speak one language.

In Qtxt you can import and export to rich text format, which the translators can then edit in Word. This solved a lot of technical issues with getting Qtxt to work in various centres around the country/globe. When they are done we then read those files back into Qtxt.

We have several modifications to the formatting of these files to make them as easy to translate as possible. One of these changes is to make inserts as clear as possible. Text in Blaise might be ‘you once told me ^name was ill’ this would become ‘you once told me [name] was ill’. Then we can tell the translators to not edit any text with square brackets around it. And because even then they will translate the odd insert by mistake, we have a checking program that check the English vs. the translation to make sure it still contains the same inserts.

So after changing the text you can click on the recompile button, and the question text will be updated within the instrument.



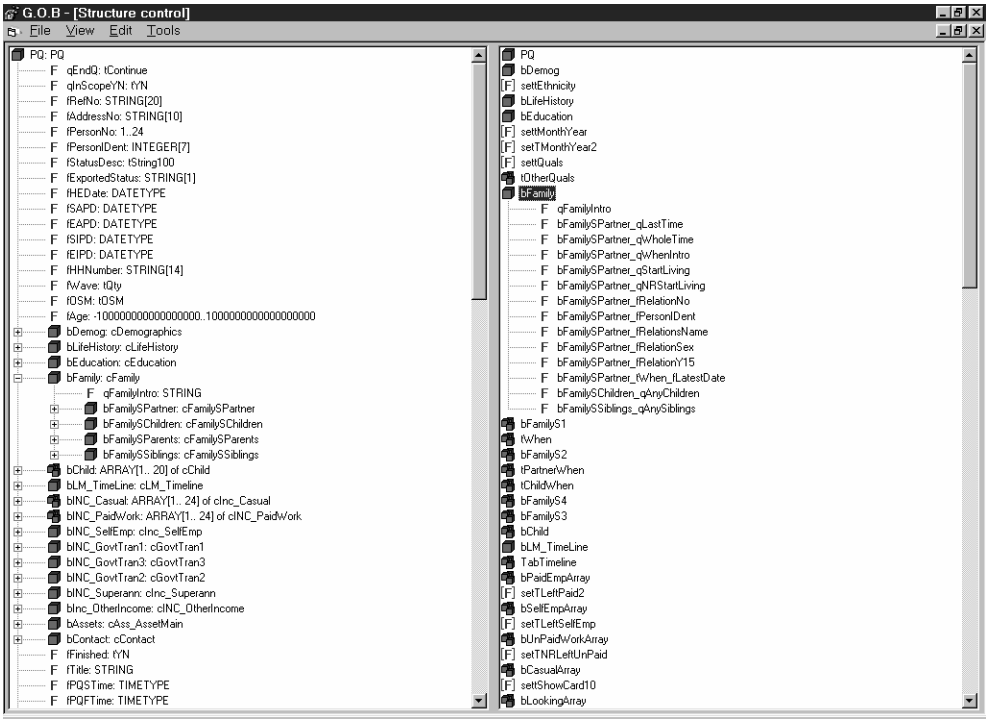
8. Appendix A- Module Manager

In Qtxt a lot of the time wasted with unnecessary text changes has been alieveated. After looking at how QDC/TAS did their design work we saw a lot of areas where an integrated design tool would help. This is how the Module Manager camne about. With it you can do anything required to design a questionnaire. You can run a module, edit its question text, open up its flow charts, and even compile the code. This tool is mostly used by QDC, but my vision is that TAS developers also use it.

9. Appendix B- B.O.S. (Blaise Output System)

BOS is the most technically challenging Visual Basic program we have written that utilises the Blaise API. The problem of how we get data out of Blaise has been one that has been at SNZ as long as I have been there. Every questionnaire we have developed has seems to try and devise a new approach to getting the data out of a questionnaire with least amount of effort.

Being the type of person who does not accept doing the same job twice I came up with BOS. What it does is load a Blaise map into a window, you can then generate a normalised view on how BOS will output the data. Then by cutting and pasting you can move fields around, rename them, delete them, etc. When you are happy with how the output tables are going to look, then you can create those tables, and populate them with data.



10. Appendix C- Where from here

So far we have managed to do some indervidual projects that have helped to make streamline some of the processes that have proved to be cumbersome in the past. This year we are looking at the whole survey design process, and how it all fits together. Hopefully after our investigations SNZ will be able to further streamline more of our development processes.

Where I see a lot of room for improvement are in the Module Manager utility. What I want to see here is a management tool that will be able to keep track of all aspects of a questionnaire development, as well as being a tool that allows access to the development environment. Some areas for improvement include.

- keeps track of code changes.
- maintain backups
- easier access to code edit information
- maintain a milestone date list for modules
- keep track of who is editing modules.
- manage rights of users

11. Appendix D- Generation of questionnaire code

There has been a lot of interest at SNZ in a project I proposed some time ago to generate code (possibly Blaise) from our flowcharts. The technology is definitely there to do this. Our tool flowcharter 2000 has a VBA backend that allows much to be done with the data stored in a flowchart. It is a direction that we would like to head, but I see it as more an icing on the cake rather than an ultimate solution. To generate questionnaire code from flowcharts requires that all questions adhere to certain standards. The most important thing is that you have those standards in place. Once we have all of our question and module types defined then life becomes that much easier for us. Generating the code from our flowcharts then becomes a logical next step.