The ScriptWriter Tool - An Application for Interviewer Training Script Development

Youhong Liu, University of Michigan

1. Introduction

ScriptWriter is a VB.net program that facilitates the creation of interviewer training scripts. The application allows the user to enter training annotations while progressing through a stand-a-lone version of the Blaise instrument. ScriptWriter produces .HTML output of the training annotations combined with the survey question text, category listing and responses that the user has entered. The output includes only the user's path through the survey instrument, versus all survey items, which is ideal for producing an interviewer training script. The .HTML output can easily be imported into Microsoft Word for further editing, as necessary, and saved as a Word document. The .HTML output, however, is pre-formatted and "ready to go" with minimal editing required.

2. History

Script development can often present a challenge for those involved. The difficulty of its production is primarily technical. Traditionally, script writers have hand-entered information from the Blaise screen. This consumes significant amount of time and staff resources, while increasing the likelihood of inaccurate and inconsistent entry by multiple writers. The need for a more efficient, automated approach to script development led to the creation of an application, called ScriptWriter.

3. Programming Language and Backend Database

The programming language used for the system is Visual Basic .net (VB.net). Previously, several applications were written with VB6. VB.net evolved from VB6 is a full object oriented programming language that offers several benefits, such as easy code maintenance, extensibility, and code reuse. These benefits are not found in procedural programming languages. The system uses Microsoft Access database to store and process data. Access is portable and easy to use for end users without significant training.

4. What's In the Program

In the ScriptWriter Programming directory:

- BlaiseScriptWriting.exe – Main program
- DEPSCRIPTGEN.exe – An interface called by Blaise DEP to input script notes
- Other Necessary DLLs – BCP DLLs, etc.
- Scripting.mdb – An Access database template

5. Data Model Files and Requirements

Required data model files:

- Blaise data model files – BMI, BXI and BDM. The core file for the system is the adt file that is generated during the data entry sessions. In order to obtain this file, the data model should be prepared with "Make Audit Trail Option" checked in its mode library.

- Blaise database files – BDB and other related files. It should at least be preloaded with primary keys. Some data models may require more involved preload (i.e. preload variables that drive the flow through the instrument).

- Blaise menu file – BMF. In this file, an item should be set to access DEPSCRIPTGEN.exe; e.g. making F6 as its shortcut.

- Other Support files – Lookup Blaise databases, DLLs and other files required by the data model

6. System Logic

6.1 Generate adt and mdb File

The first step to create a training script is to start entering a case in the Blaise DEP with a selected data model and preloaded database – See Figure 1. The access database is used to store training notes. The template scripting.mdb that is located in the application directory is copied to the data model directory when a new case is started in the Blaise DEP. It is renamed as [PrimaryKey].mdb.

During data entry, if the user wants to enter training notes, they can press "F6" to bring up the DEPSCRIPGEN program interface (Figure 2). The DEP will pass the correct field name to the program. When "Save" is clicked, it will save the data to the mdb database.

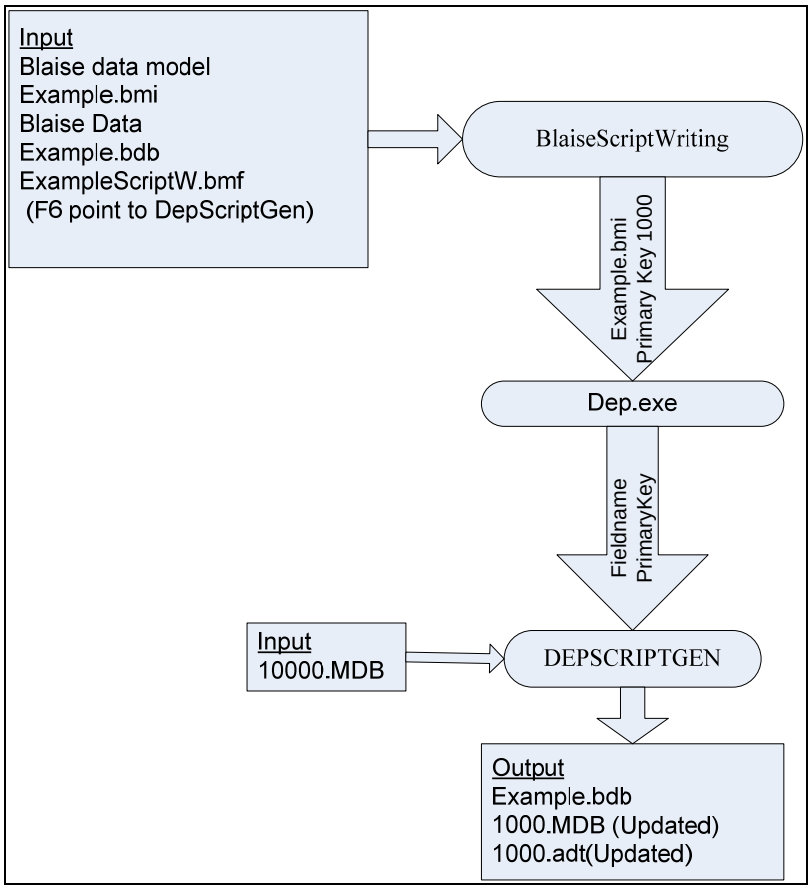

Figure 1: adt and mdb Files Flow Chart

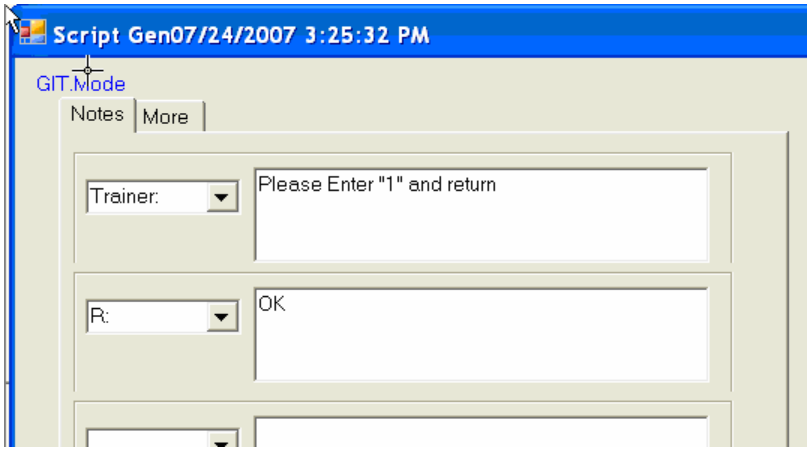

Figure 2: DEPSCRIPTGEN Interface

6.2 Editing Training Notes via ScriptWriting Program

BlaiseScriptWriting has a similar interface as that of DEPSCRIPTGEN. See Figure 3. On the left, the field sequence from the adt is listed. The field tag, name, value and code are also displayed for reference. To edit training notes for a field, just navigate to the field and update the data on the right. This way, the user can enter or edit data outside Blaise. This is important, because if the Blaise DEP is used to launch the DEPSCRIPTGEN every time the user wants to change notes for a field, the user will have to go back and forth to search for the field. It is not only time consuming, but also would leave a lot of unwanted entries in the adt file that would make the output processing more difficult.

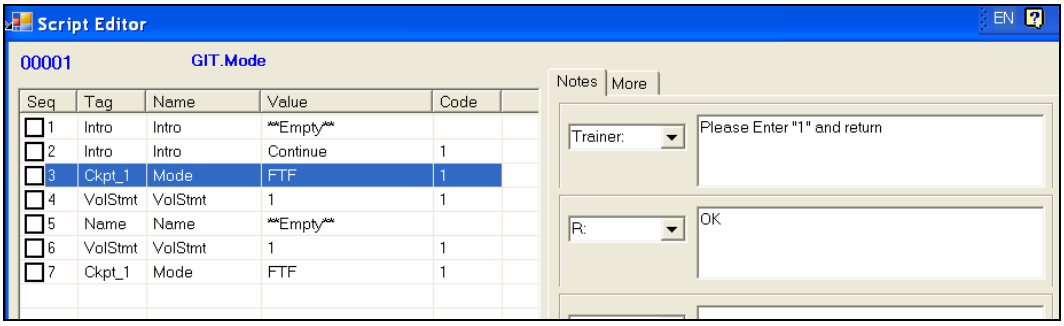

Figure 3: Editing Script Outside of DEP Session

6.3 Generating Output Html File

After the notes are all entered, the updated Access database and adt file are used to create the final script file, namely [PrimayKey].html. In Figure 5, the data items in the html file are from different sources.

Examine the contents of the file for accuracy from the html output. If necessary, the user can go back to the interface, in Figure 3, to determine if additional training notes are needed or changes required. Repeat this process until the output is satisfactory.

During html generation, a temp bdb is created to store data read from the adt file. This temp bdb is used to obtain correct question and category text. For example, if there are fills for a question dependant on responses from previous questions, the bdb with updated field must be used to obtain the correct question text. The bdb is deleted upon completion of the html file.

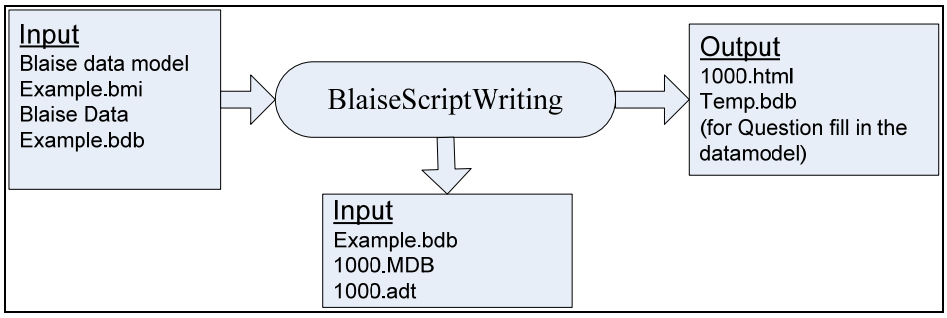

Figure 4: Generate .html Output File

(Field name from the Blaise data model)
Ckpt_1 GIT.Mode
(Field text from the Blaise database)
☑ Interviewer checkpoint:
Is this a face-to-face or telephone interview?
(Code categories from the Blaise database)
1. **Face-to-face interview**
5. **Telephone interview**

(Training notes from the Mdb file)
Trainer: Please Enter "1" and return
R: OK
(Field Value from the adt file)
Value: 1 (Face-to-face interview)

Figure 5: Sample html output

6.3 Removing Duplicate Variables from Output

Due to the nature of the ScriptWriter program, the user will need to perform a small amount of editing to account for the presence of duplicate variables either from ScriptWriter's digestion of the .ADT file, or from the user backing up to make corrections during the Blaise entry. The Seq# box of any variable can be checked to eliminate it from appearing in the script output (see Figure 6 below).

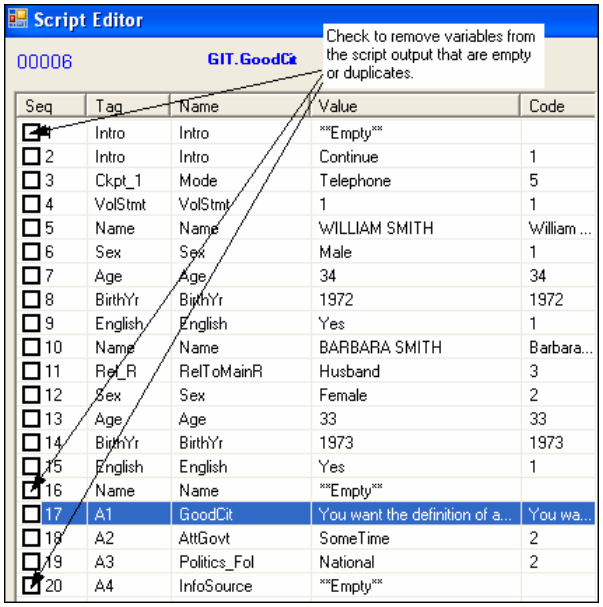

Figure 6: Removing Duplicate Variables

6.4. Script Lite

The reason to add the ScriptLite function to the system is to deal with data model changes. For instance, if fields are added to the data model and the user continues to use the adt produced by the previous data model, the output will be out of sequence (assuming the fields were added somewhere in the middle of the instrument). In this case, the whole case has to be reentered to ensure that the correct questionnaire flow sequence is in the adt. To preserve the data entered in the previous data model, the ScriptLite is used to save that data to the mdb so the user can quickly go through the case in the new data model. Note that at this point, the training notes the user spent hours inputting via the previous data model in the mdb are still available. Only training annotations for new questions needed to be added. The data is saved to tblScriptList table. A useful report in the database is designed in the mdb so users can print out the content to help them to reenter the data. See Figure 7.

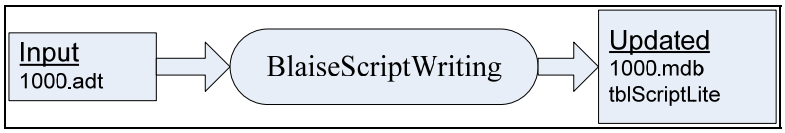

Figure 6: Script Lite Flow Chart

tblScriptLite

Seq	FieldName	FieldTag	Code	CodeText
2	GIT.Intro	Intro	1	Continue
3	GIT.Mode	Ckpt_1	5	Telephone
4	GIT.VolStmt	VolStmt	1	1
5	GIT.HHL.HHL[1].Name	Name	Joe Smith	JOE SMITH
6	GIT.HHL.HHL[1].Sex	Sex	1	Male
7	GIT.HHL.HHL[1].Age	Age	34	34
8	GIT.HHL.HHL[1].BirthYr	BirthYr	1972	1972
9	GIT.HHL.HHL[1].English	English	1	Yes
18	GIT.Intro	Intro		Continue
25	GIT.GoodCit	A1	Someone who votes in the elections and is aware of issues at the national and local level.	Someone who votes in the elections and is aware of issues at the national and local level.
26	GIT.AttGovt	A2	2	SomeTime
27	GIT.Politics_Fol	A3	1	International
28	GIT.InfoSource	A4	2	Magazines
29	GIT.Comp_Use	B1	5	No

Figure 7: ScriptLite Output

6.5. Importing the .HTML script Output into Microsoft Word

Finally, after users confirm that they have everything required from the ScriptWriter, the html file can be imported into Microsoft Word for further editing. The users can edit the document and adjust formatting as necessary. In some cases, the margins of the page will need to be widened.

6.6 Translate Script Notes to the Other Language

In the mdb, form LanguageTran is designed to translate the training notes to the other language, see Figure 8. In the ScriptWriter interface, select the data model language along with the "User Other Language" check box, as seen in Figure 9.

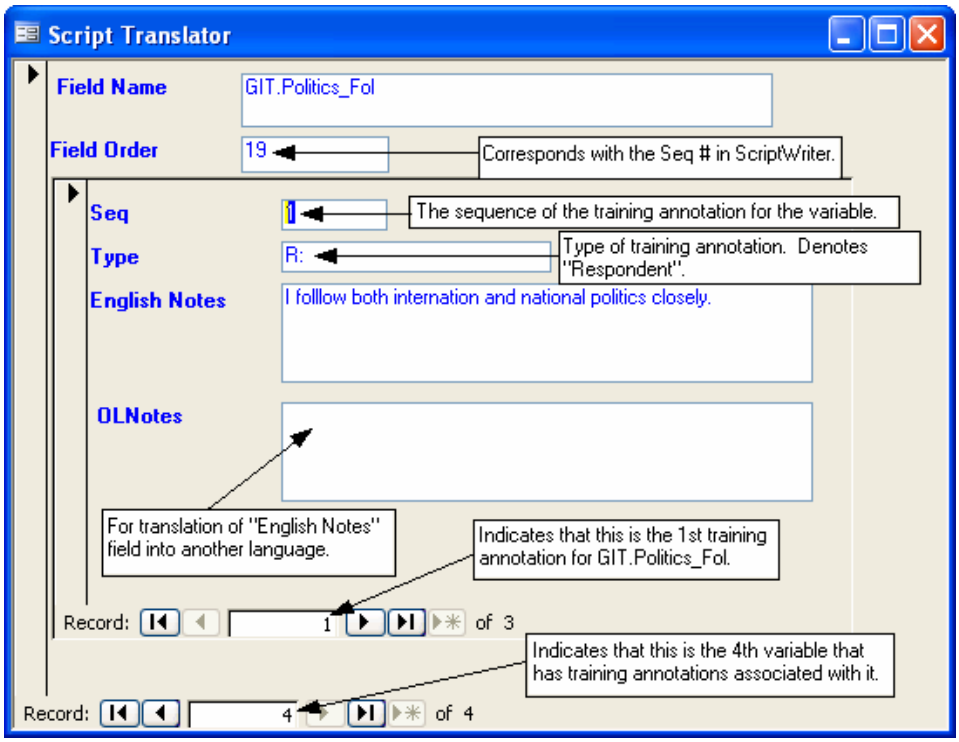

Figure 8: LanguageTran Form

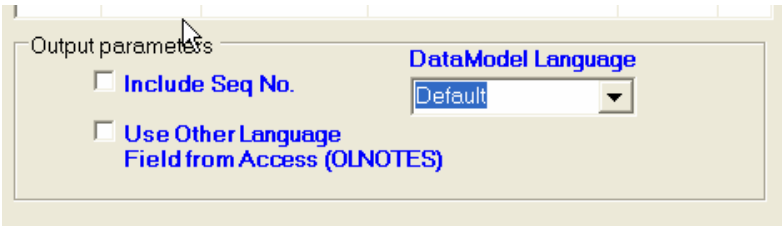

Figure 9: Output Parameters

7. Randomization in the Blaise Datamodel

If the Blaise data model has a randomization field, and some logic and question fills depend on the value of this variable, problems may occur when using the system. This is because Blaise may generate a different value during time of script generation than it generated during time of training. In this situation, questionnaire flow and question text maybe different from computer to computer. To avoid this, it is advised to preload randomized values.

8. Conclusion

Interviewer training scripts play an integral role in Computer Assisted Interviewing (CAI) studies. They are used in interviewer trainings to provide them with practice through a Blaise questionnaire, while trainers highlight important features. Scripts are also used to assess interviewer questionnaire administration, to ensure the quality of data and to offer a means to standardize training elements. Standardization is crucial to the proper training and certification of interviewers, especially across training teams with multiple training sessions.

The ScriptWriter Tool was developed to facilitate the script development process. The goal was to create a user-friendly application with the flexibility to serve the needs of many projects and diverse Blaise applications. In most cases, using ScriptWriter should result in significant time and resource savings in the production of interviewer training scripts, as compared with alternative methods. Since the question components are pulled directly from the Blaise instrument, the resulting script is accurate and contains the components necessary to ensure the proper training and certification of interviewers.