The Use of Metadata in the Design of a Customizable .NET Event History Calendar

Daniel Moshinsky, Mecene Desormice, Seth Benson-Flannery; U.S. Census Bureau, USA

1. Overview

The Survey of Income and Program Participation (SIPP) is a large, long-standing longitudinal survey that collects sources and amounts of income, labor force information, program participation and eligibility data, and general demographic characteristics.

SIPP is undergoing a re-design. As a part of the re-design, it was decided to increase the reference period for the survey from four months to twelve months of the previous calendar year. An Event History Calendar (EHC) is used to help address the concern that the longer reference period may impact respondent recall.

About half of the SIPP questions are asked in the Blaise 4.8 DEP, while the other half are asked in the EHC module, which is written in C# .NET, and invoked from the DEP as a COM object DLL.

This paper focuses on the use of external metadata files and the Blaise 4.8 API to facilitate communication of data and metadata between the Blaise database and the EHC and the role of external metadata in the dynamic generation of a fully customizable Event History Calendar. Advantages, details of implementation, and applications of this approach are discussed.

2. Design and organizational considerations

Several critical survey and organizational considerations guided our selection of a design approach. These considerations are described below.

2.1 Data collection methods

SIPP contains some 40 sections, such as sections relating to employment history, health insurance coverage, assets, utilization of services and demographic history. Not all of SIPP questions are well-suited for being asked in the EHC.

It has been shown that the Event History Calendar method of interviewing is more effective than the standard question list method in improving recall of dates of autobiographical events (Belli et. al., 2001). Therefore, the types of SIPP sections that are asked in the Event History Calendar contain questions involving recall of dates of multiple changes in status throughout the reference period. For example, questions about the dates of job changes or changes in receipt of benefits are collected through the EHC.

On the other hand, questions relating to assets pertain to the reference period as a whole (e.g. “What was the combined value of your assets in the previous calendar year?”), and do not involve recall of dates. Thus, those sections can still be asked in the standard question list method, and can be programmed in Blaise.

2.2 Data exchange

Another requirement is that sections throughout SIPP must be able to exchange data – to pass parameters to each other. For example, EHC sections must be able to use the household roster collected earlier in Blaise, and a commuting module in Blaise must be able to use employment information that had been collected in the EHC. For this reason – and to simplify post-collection data processing – all survey data is stored in a single database.

2.3 Fluid requirements

Since the use of the electronic EHC is new to SIPP and the US Census Bureau, it is understood that requirements may change significantly throughout the development cycle. As the software undergoes internal and field testing, new questions and sections may be moved in and out of the EHC, wording and composition of individual questions may change, skip patterns may change, and so on. Thus, the design of the EHC must be flexible enough to rapidly accommodate changing requirements.

2.4 Large number of questions in the EHC

There are twenty five sections in the SIPP EHC, each with a different number of potential follow-up questions. For example, once the respondent indicates having been employed from January to September, some forty questions (depending on the skip patterns) will be asked about that spell of employment.

Blaise programmers are accustomed to being able to rapidly program fields and skip patterns -- taking advantage of much behind-the-scenes processing being powered by the Blaise software. Such luxury is obviously not available to programmers working on the EHC in a language other than Blaise – such as a .NET-based language. A lot of code must be written in .NET to correctly place even one question on the screen, to lay out its answer list, to implement an edit check, apply a skip pattern, or to store and retrieve data. With over a hundred fields in the EHC, a more streamlined approach was needed to make the initial programming and subsequent maintenance feasible.

2.5 Re-use of the EHC

Event history calendars are a promising emerging data collection technology. If the integration of the EHC into the SIPP survey is successful, then it is possible that some other surveys may follow suit. The ease with which the EHC can be integrated into a new survey was an important consideration in our design choices.

2.6 Organizational knowledge

Since the majority of surveys developed by our group are programmed in Blaise, there is not a lot of knowledge among the staff of C#. It was important to make it as simple as possible for a Blaise programmer to maintain the EHC module without knowledge of C#, and even to enable such a programmer to customize the EHC for use in a new survey with minimal changes to the underlying C# source code.

In other words, it was important to leave the underlying processing transparent to the survey programmer – much like it is in Blaise.

3. Design Overview

In a nutshell, the SIPP EHC is designed as follows. All sections and all questions that are asked in the EHC are defined as blocks and fields in the Blaise survey datamodel. No fields are hard coded in the C# EHC code. Instead, the C# code includes what can be thought of as templates-processing code. It dynamically generates questions and places them on the screen based on attributes of the metadata – such as question type, question text, answer list text, field length, flow (skip) conditions, and the like.

Three external metafiles – “Questions”, “Topics/Sections”, and “Rules” – are used to drive dynamic generation and placement of fields in the EHC. These metafiles serve as links between the fields defined in the Blaise database (accessed via the API) and the .NET system that displays them.

3.1 Blaise Database

One unified standard Blaise database is used for the entire data collection instrument. All blocks, fields, answer lists, and values that appear in the EHC are contained in the Blaise database and accessed via the Blaise API.

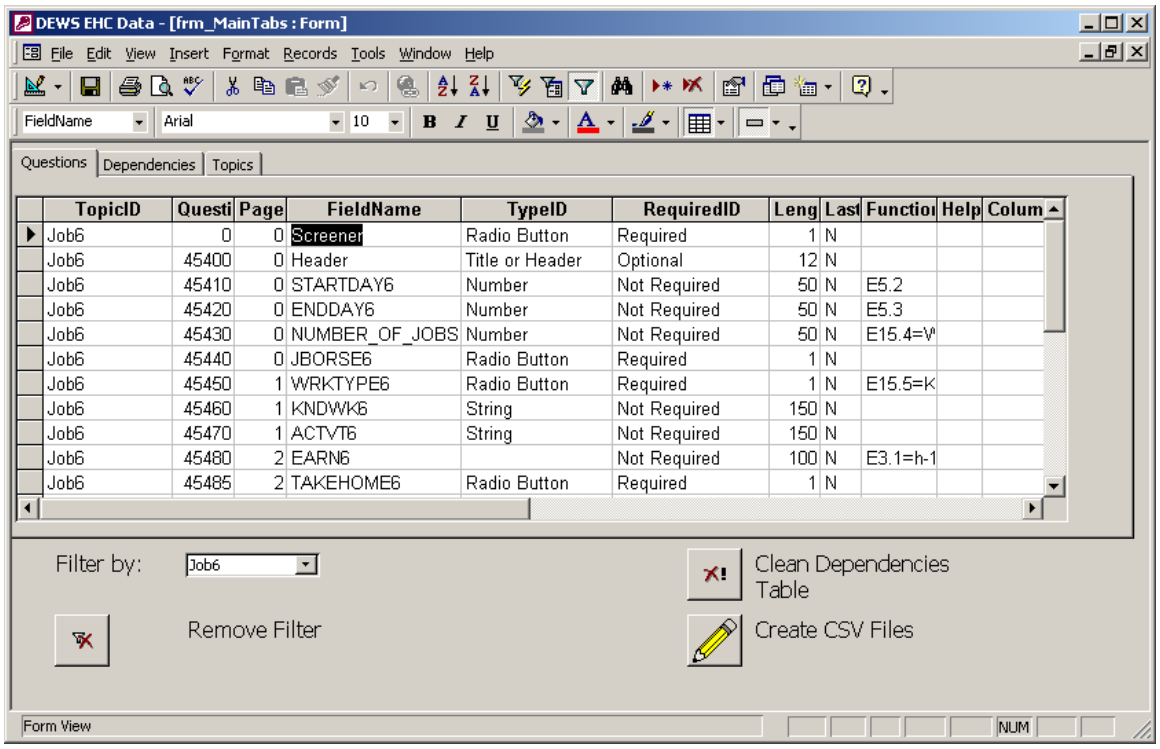
3.2 External metafiles

3.2.1 “Questions” metafile

This metafile describes the metadata needed to display each individual field in the EHC. It includes the block to which the field belongs, the sequential order number in which the field will appear, page number onto which the question will be placed within the EHC (used the same way pages are used in the Blaise modelib), field name and type, length (for text fields), and whether the field allows an Empty value or not.

In addition, this metafile allows a “function” to be attached to a field, and lets the C# system know that additional processing is needed either prior to displaying this field or after a user enters a value. Examples of additional processing are complex skip patterns, edit checks, or dynamic answer lists. Functions are discussed in greater detail below (Section 3.3.2).

The Questions metafile also includes a column to indicate to the system that a help screen is associated with a question. The C# system uses this flag to display the question-specific help screen when the user presses the F1 key while positioned on the question.



TopicID	Questi	Page	FieldName	TypeID	RequiredID	Leng	Last	Functioi	Help	Colum
Job6	0	0	Screener	Radio Button	Required	1	N			
Job6	45400	0	Header	Title or Header	Optional	12	N			
Job6	45410	0	STARTDAY6	Number	Not Required	50	N	E5.2		
Job6	45420	0	ENDDAY6	Number	Not Required	50	N	E5.3		
Job6	45430	0	NUMBER_OF_JOBS	Number	Not Required	50	N	E15.4=V		
Job6	45440	0	JBORSE6	Radio Button	Required	1	N			
Job6	45450	1	WRKTYPE6	Radio Button	Required	1	N	E15.5=K		
Job6	45460	1	KNDWK6	String	Not Required	150	N			
Job6	45470	1	ACTVT6	String	Not Required	150	N			
Job6	45480	2	EARN6		Not Required	100	N	E3.1=h-1		
Job6	45485	2	TAKEHOME6	Radio Button	Required	1	N			

Figure 3.2.1 The “Questions” metafile

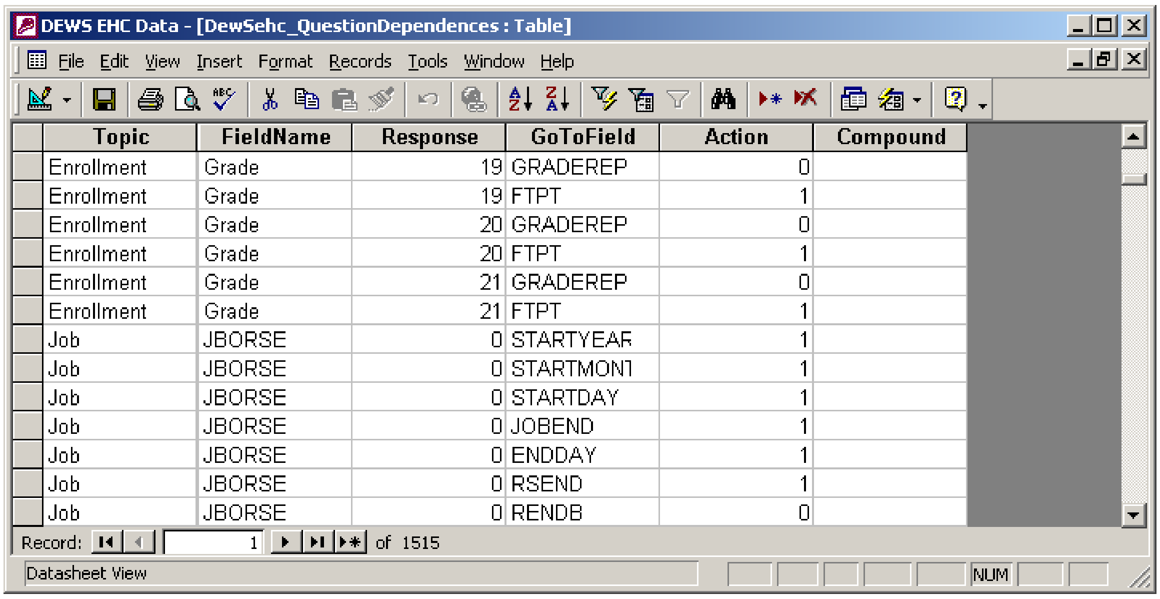
3.2.2 “Rules” metafile

Another metafile is used to describe the rules – i.e. the conditions for when a question is brought on-path or off-path based on answers to preceding questions. This file allows for implementation of simple conditional statements without writing any code in C# -- the person entering the rules must only enter the question ID of the antecedent question and its value (i.e. “**if** question 30 is Yes”), as well as the question ID of the consequent and its on- or off-path status (i.e. “**then** question 31 is Off-Path”).

In the screenshot below (Figure 3.2.2), the antecedent is described in column FieldName, and the consequent is described in column GoToField. A value of 0 in the Action column indicates that the GoToField is off-path for the given response to FieldName, and a value of 1 indicates that it is on-path. In the screenshot below, if values of 19, 20, or 21 are entered in field GRADE, then the GRADEREP field is off-path, but field FTPT is on-path.

The “Compound” column allows for an “AND” combination of more than one value to determine status of the GoToField.

This metafile handles the majority of skip patterns in a questionnaire. Some questions, however, require a more complex skip pattern than a simple AND/OR conditional. For instance, values from outside the block may be considered, or values may be combined in a complex way. Handling of more complex rules like these is done through Functions (see section 3.3.2 below).



DEWS EHC Data - [DewSehc_QuestionDependences : Table]

Topic	FieldName	Response	GoToField	Action	Compound
Enrollment	Grade	19	GRADEREP	0	
Enrollment	Grade	19	FTPT	1	
Enrollment	Grade	20	GRADEREP	0	
Enrollment	Grade	20	FTPT	1	
Enrollment	Grade	21	GRADEREP	0	
Enrollment	Grade	21	FTPT	1	
Job	JBORSE	0	STARTYEAR	1	
Job	JBORSE	0	STARTMONT	1	
Job	JBORSE	0	STARTDAY	1	
Job	JBORSE	0	JOBEND	1	
Job	JBORSE	0	ENDDAY	1	
Job	JBORSE	0	RSEND	1	
Job	JBORSE	0	RENDB	0	

Record: 1 of 1515

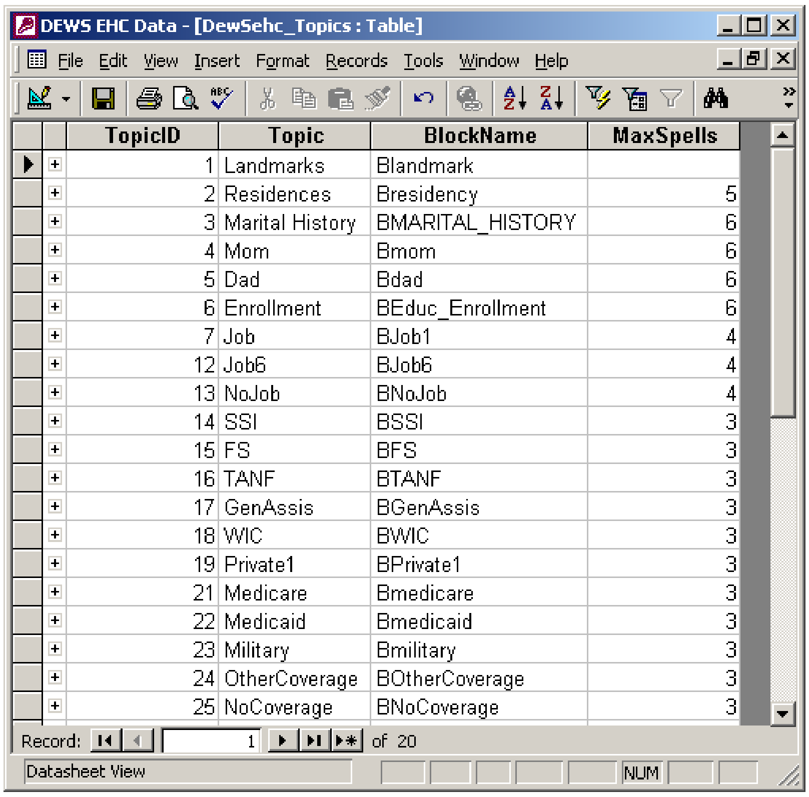
Datasheet View

Figure 3.2.2 The “Rules” metafile

3.2.3 “Topics/Sections” metafile

This file describes the metadata of the instrument at the section level. It consists of a table of topic IDs – that determines the order in which topics are displayed in the EHC – and topic names corresponding to block names in the Blaise datamodel. An additional column describes the maximum number of spells each topic can have, as requested by the specifications (due to the great number of blocks in this person-based survey and the size limitations of the Blaise database, it was necessary to limit the maximum number of spells). For example, no more than 5 changes of residence during the reference period may be recorded for each household member. If the user attempts to enter a 6th residence, an edit check appears.

Additional topic-level attributes (perhaps a topic-level help screen or edit check) may subsequently be included in this table, if needed.


DEWS EHC Data - [DewSehc_Topics : Table]
File Edit View Insert Format Records Tools Window Help

TopicID	Topic	BlockName	MaxSpells
1	Landmarks	Blandmark	
2	Residences	Bresidency	5
3	Marital History	BMARITAL_HISTORY	6
4	Mom	Bmom	6
5	Dad	Bdad	6
6	Enrollment	BEduc_Enrollment	6
7	Job	BJob1	4
12	Job6	BJob6	4
13	NoJob	BNoJob	4
14	SSI	BSSI	3
15	FS	BFS	3
16	TANF	BTANF	3
17	GenAssis	BGenAssis	3
18	WIC	BWIC	3
19	Private1	BPrivate1	3
21	Medicare	Bmedicare	3
22	Medicaid	Bmedicaid	3
23	Military	Bmilitary	3
24	OtherCoverage	BOtherCoverage	3
25	NoCoverage	BNoCoverage	3

Figure 3.2.3 The “Topics/Sections” metafile

3.3 The C# system

3.3.1 Dynamic processing

All sections and fields are generated and displayed at run-time by querying the Blaise API and the metafiles described above. The C# code contains implementations of generic methods for each operation necessary to display blocks, fields, and answer lists of every type (e.g., string, numeric, radiobutton, checkbox, dropdown), as well as methods to process the rules, display edit checks, retrieve and store data, and so on.

3.3.2 Functions

More complex event processing is handled through “functions” -- methods written in C#. The “Questions” metafile described in Section 3.2.1 contains a column with a function number (e.g., “E5.2”) that associates one or more methods with a particular field. When a field loads, or when a value is entered in it, the system “sees” that the field has an associated function (or functions), and the system proceeds to process the instructions in that function.

A function may perform one of several jobs. It may be called upon to process an edit check, determine a skip pattern, populate a dynamic answer list, or perform some other specialized role.

The skip patterns relegated to the functions are ones that are too complex for the Rules metafile to handle. Still, the programming involved in writing a new function is fairly straightforward and requires very little knowledge of the C# language beyond familiarity with the syntax of writing conditional statements.

Most edit check processing functions are generic and shared by many fields. For example, the range check function – which checks to see that a numeric value falls within a specified range – is used by nearly every numeric field in the EHC. The re-use of functions across fields is accomplished by different parameters being associated with different fields. Parameters include whether the check is hard or soft, and may include ranges of valid values, and so on.

For example, the field EARN6 (a question about earnings) in Figure 3.2.1 has a function associated with it. The full value in the Functions column for that row reads “E3.1=h-1000-999999”. The letter “E” in front of the function number indicates that the function will execute following an Entry in that field. The number of the function is followed by the letter ‘h’ – indicating an instruction to display a hard check if a value is entered that is either less than 1,000 or more than 999,999).

Once the generic edit check and skip pattern functions have been programmed, they can be re-used by many other fields in the instrument simply by passing a different set of input parameters to the functions. As a result, all numeric fields can share the same range check function, and skip pattern functions can be shared by passing names of involved fields as parameters.

The functions feature of the design allows for sophisticated error and flow handling within the EHC, while keeping the EHC easy to maintain and customize for Blaise programmers unfamiliar with the advanced features of C#.

4. MS Access utility

During initial development, the metafiles were being maintained as comma-delimited text files and modified either directly with a text editor, or via Excel. As development continued, and the number of questions implemented in the EHC grew, so did the size of the text files, and it became difficult to maintain the files manually.

In order to build a more reliable and maintainable application, we then developed a small MS Access database application to organize the information. This tool provides a way to ensure consistency of the data entered by performing some basic checks on the validity of entries and using dropdown lists to restrict possible values. It allows us to maintain, update, and modify the metadata more accurately.

5. Examples of metafile use

5.1 Adding a question

To add a question, one would first add a field to the correct block in Blaise, and then simply add a line to the Questions metafile table, filling all required cells.

5.2 Re-ordering questions

To change the order in which questions come up (e.g., to reverse the order in which State and County are asked), one would simply modify the QuestionNumber column in the Questions metafile.

5.3 Changing question types

To modify the type of the question (e.g., change a numeric field to a text field), one would simply change the value of the Type column in the Questions metafile.

5.4 Changing question or answer list wording

In order to change the wording of a question or an answer category, one would modify them in the Blaise datamodel and rebuild the Blaise project. The EHC would pick up the new wording because the EHC retrieves the wording from the Blaise datamodel via the API.

5.5 Adding a new EHC section

In order to add a new section to the EHC, first a new block needs to be defined in Blaise. Then, a line must be added to the Topics/Sections metafile. Subsequently, questions and rules (if any) must be entered in the other two metafiles.

6. Conclusion

The three-level interaction among the EHC .NET code, the Blaise API, and the external metafiles yields a robust data collection system that not only includes all the features of an Event History Calendar, but also nearly the same question flow and error checking that our users have come to expect from Blaise surveys.

Almost all C# processing is transparent to survey programmers who need to only concern themselves with entering the metadata into Blaise and into the external metafiles, and perhaps also with modifying a few of the functions.

Specifying EHC fields in the metafiles and the Blaise database has the additional advantage that enhancements to the EHC functionality and layout can be carried out independently of decisions about specific items. Fields and rules can be modified in the metadata without affecting the datamodel – the C# DLL does not need to be re-built in order to effect the changes to the metafiles. This makes maintenance after deployment easier, and allows for an immediate fix to certain kinds of problems that might occur in the field.

The SIPP EHC system is still young and in continuous development. Many technical challenges have been overcome, and quite a few still remain. All of the core design features discussed above have been successfully implemented, but we hope to improve upon some of them in the future. For example, it would be particularly helpful to validate metadata entry and make it more user-friendly. Another possible enhancement is to import parts of the metafiles directly from specifications.

Lastly, we have encountered multiple other challenges while developing the EHC, such as issues with invoking the COM object DLL from Blaise, exchanging values through the Blaise API, cleaning off-path data in the EHC, and others. The description of those challenges lies outside the narrow scope of this paper, but we hope to report on them in the future.

7. References

Belli, Robert F.; Shay, William L., and Stafford, Frank P. . Event History Calendar and Question List Survey Interviewing Methods: A Direct Comparison. *Public Opinion Quarterly. 2001; 65*(1):45-74