Capturing Signatures Electronically from a Blaise Instrument

Todd Flannery, Ed Dolbow, Michael Kapombe, and Curtis Cooper, Westat, United States

1. Study Background and the Use of Electronic Signatures

For A Large Household Survey (ALHS) Westat interviewed several sample groups residing at the same address including adults, youths and parents. Each interview required a signed consent or assent form and led to an incentive and signed receipt at the completion of the instrument. Consenting adults provided specimens and received a second incentive. Depending on the household composition up to 8 different forms were required in a household and some households had more than one participant in each sample group. It became clear during the field test that the paper management required of the field interviewer and the paper tracking required of the home office was excessive and error prone. We devised an electronic signature approach and integrated it into the Blaise instruments to solve the problem.

We use Blaise 4.8.4 to collect data related to the consent or receipt; call a .NET C# application passing the relevant data; capture the e-signature image; and return control to the Blaise instrument. In addition, we store and encrypt a PDF file containing the statement and signature associated with the participant. Our paper will discuss the process of having Blaise 4.8 administer an electronic e-signature application for the purpose of collecting and documenting these household consents and incentive receipts, the limitations of using a shell-to-EXE application, as opposed to activation of a DLL as an alien router, and adapting the e-signature procedure to Blaise 5 for administering future surveys to this population.

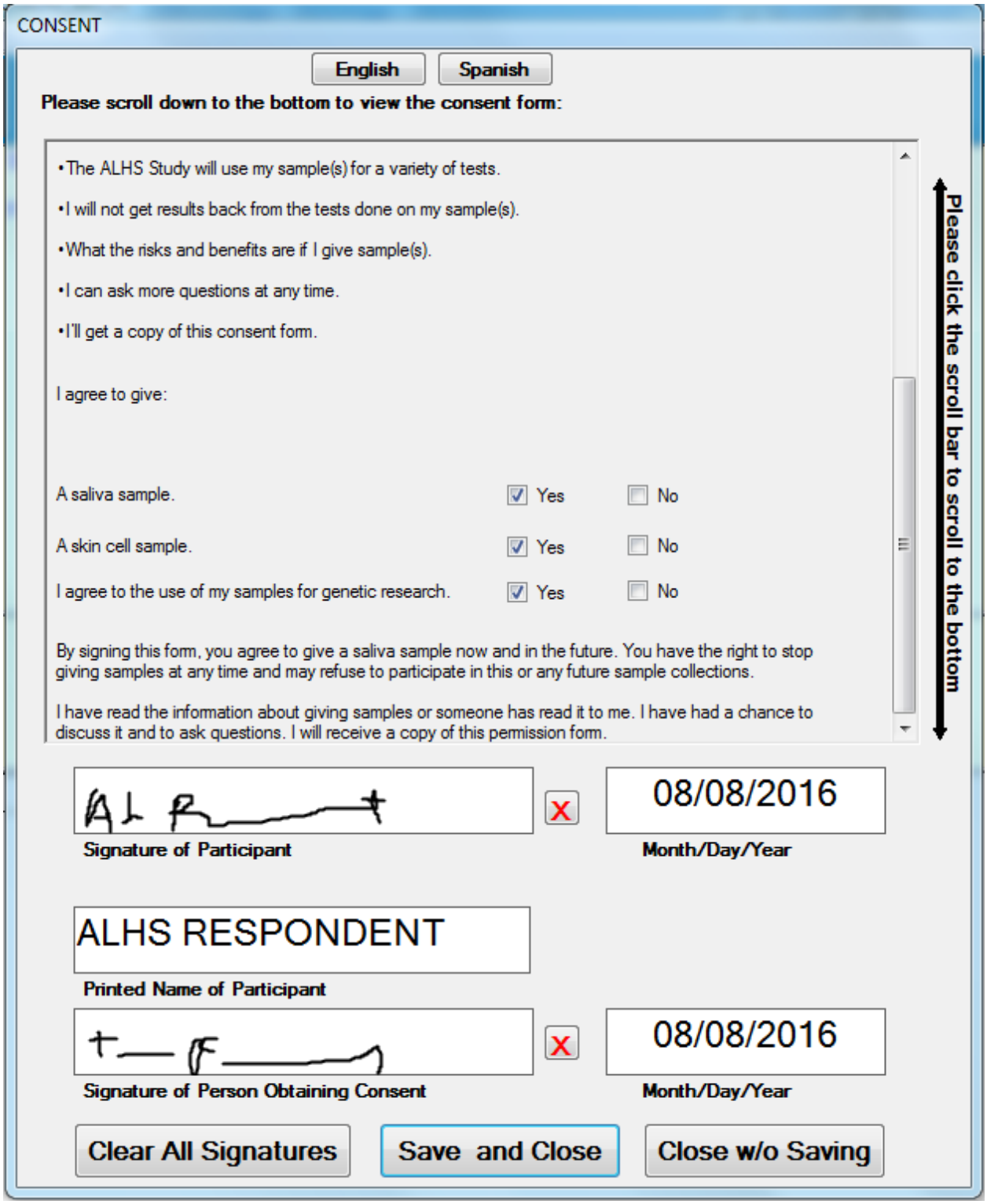
2. The Electronic Signature Form

The e-signature application is a .NET console application written using the C# language to collect signatures. A Blaise instrument validates an array of strings in the main program and launches the e-signature application, passing the array of strings as arguments. The validation process checks that important parameters are not missed before the e-signature form is shown to the respondent. This .NET application was designed mainly using forms with the following controls:

- **Label control**: display text on the form
- **Panel control**: assign frame on the form
- **Textbox control**: capture user signature. This control required using Microsoft.Ink.DLL.
- **Checkbox control**: keep track of type of sample that will be collected.
- **Picture Box control**: save the entire form as GIF picture file on a secure location where it will be encrypted and electronically transmitted to WESTAT.
- **Resource file**: store text strings that are displayed on the form. This control helps customize the E-Signature app for different kinds of signatures for various consents and incentive receipts. In addition it can also be used for images, icons, audio, and video files.

We produced an executable application, external to Blaise, to display one of several forms depending on the Blaise instrument calling the application. The form accepts several parameters that determine text displays. We also display the respondent's selections made in the Blaise Data Entry Program (DEP) as check boxes on the form (see Figure 1).

Figure 1. The .NET E-Signature Form

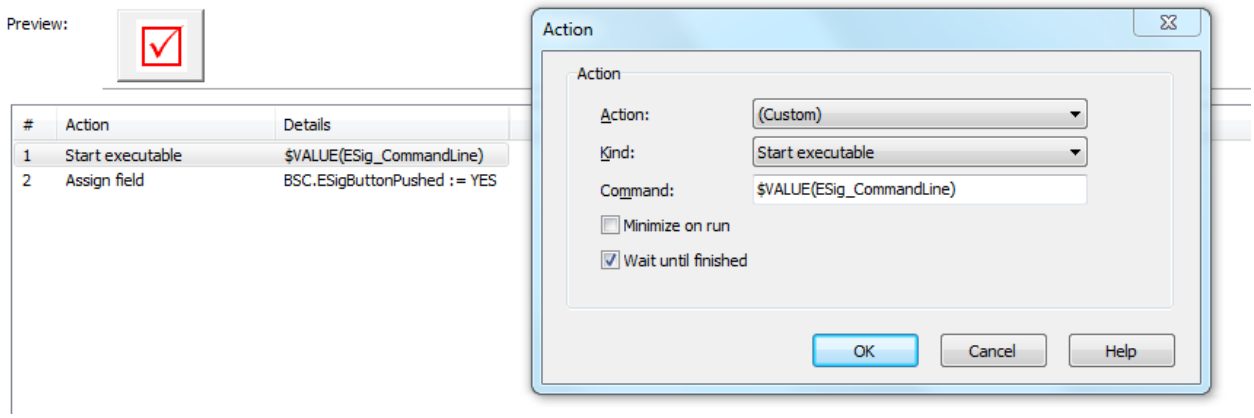


Using a stylus, touch screen, touch pad, or some other input method, the respondent chooses to accept the displayed selections, sign via the e-signature controls, and "Save and Close" or return to the DEP to alter the selections "Close without Saving". If the form is saved, the data are validated to ensure that a signature was collected, and a GIF picture file of the electronically signed form is stored in an encrypted location.

3. Modifications in Blaise 4.8

In the DEP, we shell out to our external application, pass parameters, determine what took place, and ensure that we have a saved copy of the document. To accomplish these tasks, we opted to use a customized DEP menu (see Figure 2).

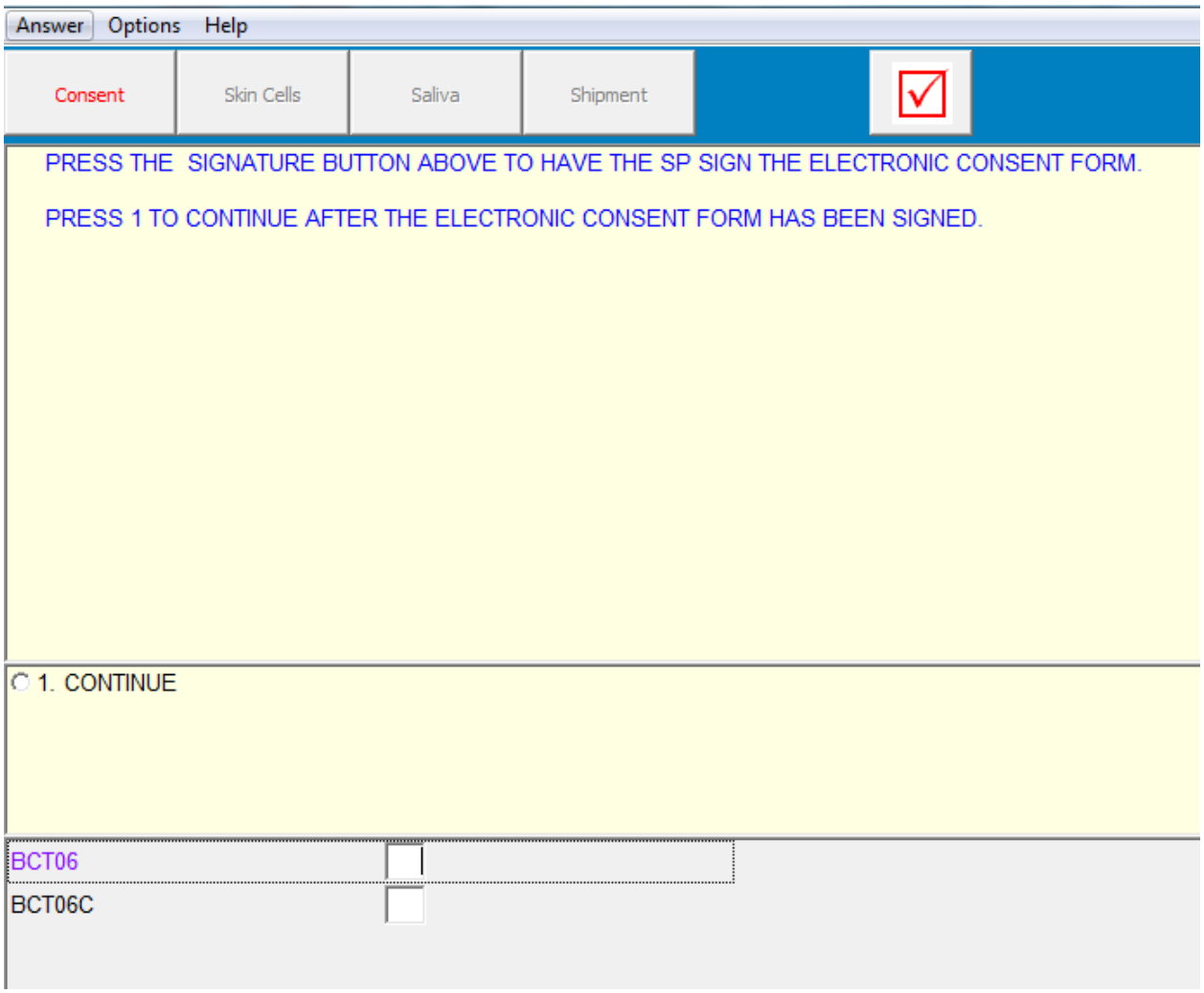
Figure 2. Settings in the DEP Menu



We define in our Panels a raised check button as a visual cue for collecting e-signatures, and we enable it when the respondent has provided the necessary consents (see Figure 3).

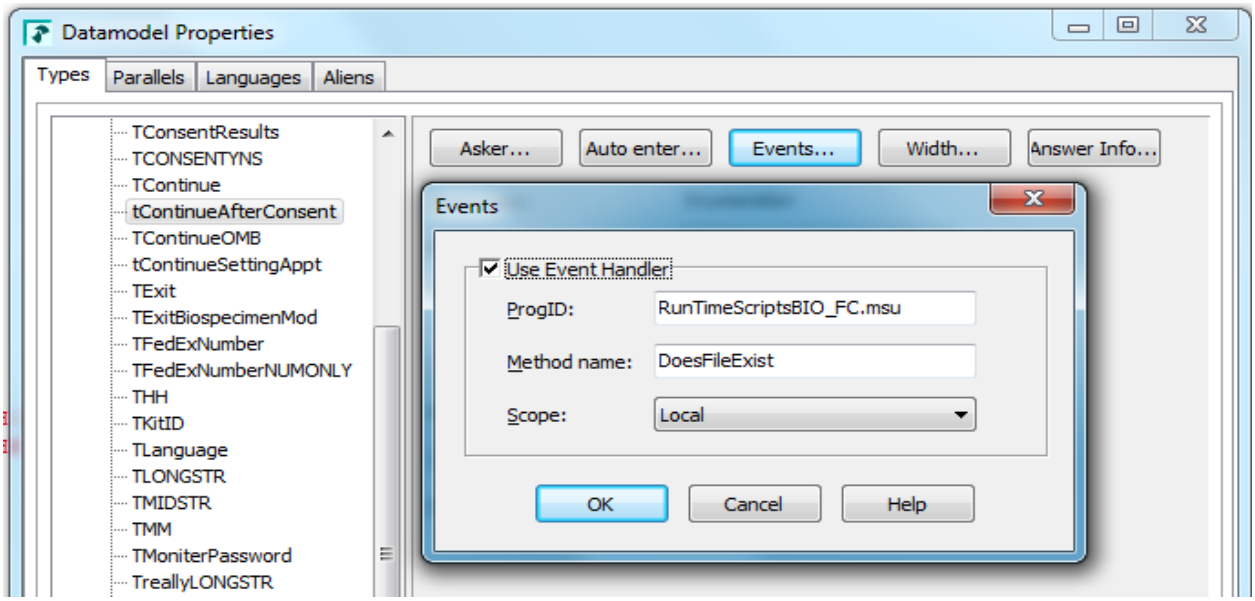
Pressing the button starts the external application, passing parameters as a string and setting the focus to the external application until the application is closed. If we are successful in obtaining the signed e-signature document, we disable the button, unless the respondent decides to change their consent in the DEP. We use a hard edit to prohibit the interviewer from moving past this screen without clicking the button.

Figure 3. Screen in DEP for Calling E-Signature Form



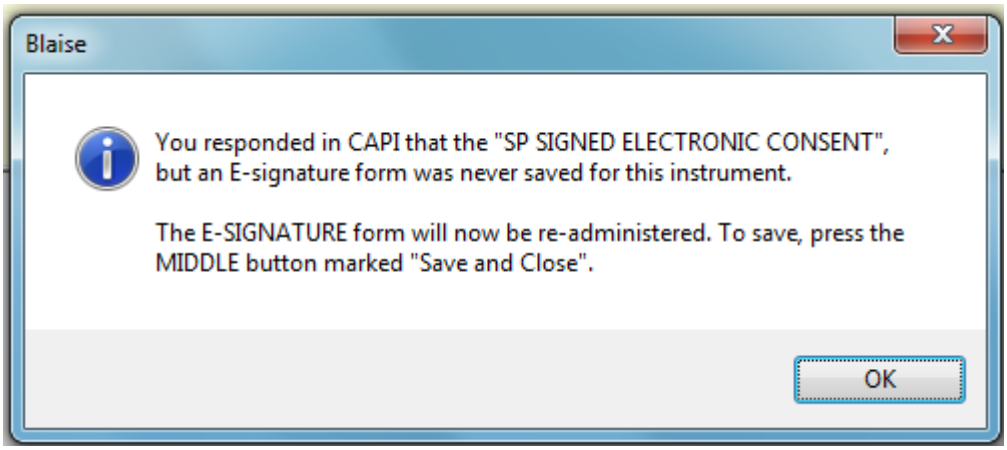
Since DEP is unaware of the result of the e-signature application, we are able to use some Maniplus scripting to determine if the result was successful by detecting that a GIF picture file exists for the captured signature. To accomplish this, we first associate a Manipula procedure as an event in the Datamodel Properties with a predefined TYPE (see Figure 4).

Figure 4. Datamodel Properties Settings



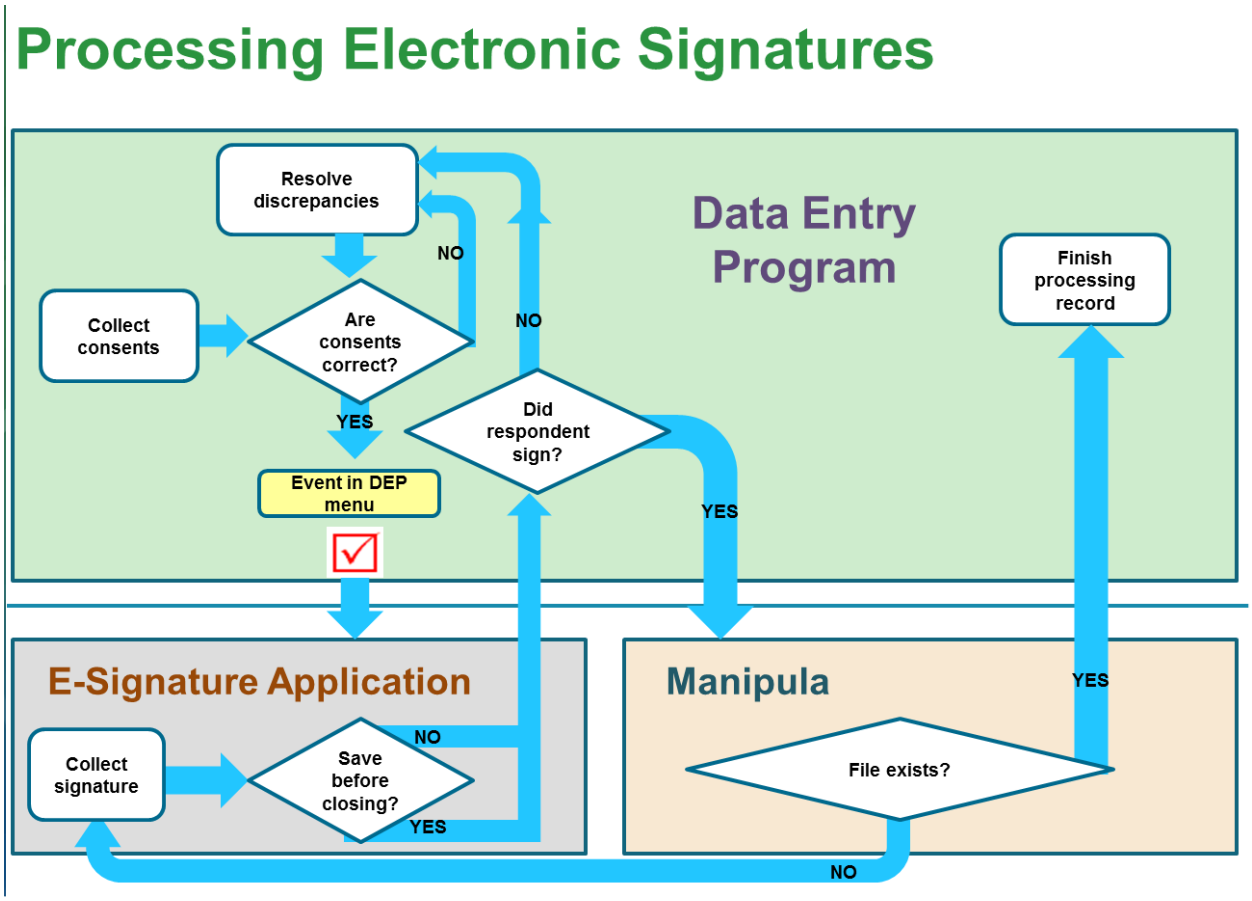
The interviewer is asked in DEP whether or not the respondent signed an e-signature form. A positive response initiates the Manipula procedure. The procedure uses the FILEEXISTS function to ensure that the image of the signed document exists. If there is a discrepancy, an error message is displayed for the interviewer via Maniplus, and the E-Signature module is re-initialized (see Figure 5).

Figure 5. Error Message When a Signed Document is not Found



The process of using an external application to collect electronic signatures is detailed in the following flow chart (see Figure 6).

Figure 6. Processing Electronic Signatures as External Application



4. Limitation of Shell-to-EXE Approach in 4.8

Since the .NET application exists externally, DEP is unaware of what is taking place in the e-signature form until after the application is closed, DEP regains the focus, and the interviewer indicates a result (signed or not signed). We could avoid asking the interviewer for a result, and perhaps displaying an error message, if the buttons on the e-signature form sent values directly to the active record in DEP.

5. Using a DLL as an Alien Router in 4.8

As an alternative approach we could develop our .NET application as a DLL and reference an alien router in the code. This approach eliminates the need for input from the interviewer after the menu button is pressed, since the alien can be initiated through logic in the RULES, and the result can be passed directly to the active record.

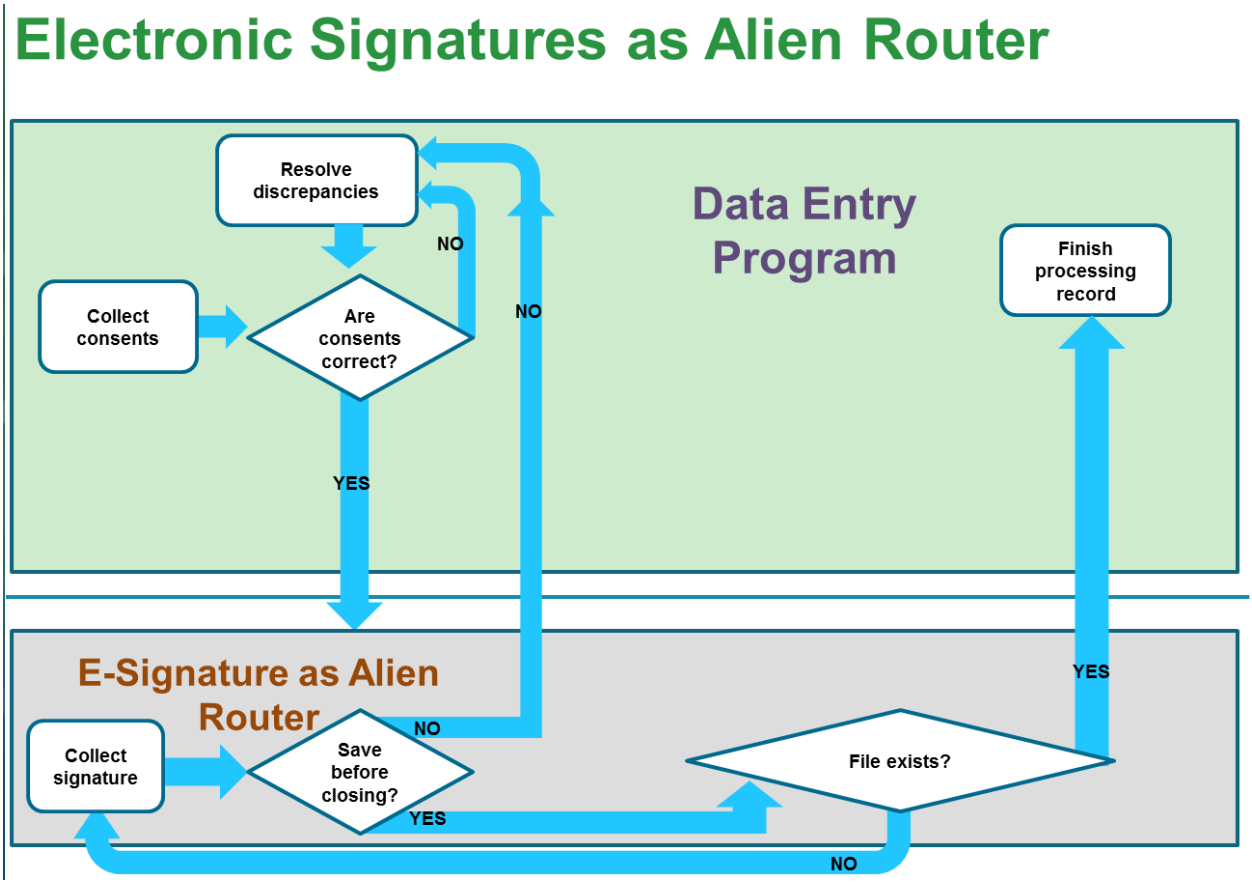
6. Converting the External Application to DLL

To convert this standalone program into a proper Blaise router, we convert it to a COM object, giving the class a GUID and a ProgId, and marking either the appropriate classes and methods or the assembly (as a whole) as COMVisible. This is the process for creating any COM object. As a standalone program, the e-signature application is called with the command line parameters via Maniplus or a button in the DEP menu file. Rewriting the program as a DLL requires that we change how the program is called. Since the look and behavior of the e-signature form as an external application depends on these command line parameters, this poses a significant problem. To replace this interface we would create a method for every e-signature form type that we need and create string auxfields in the router block for any dynamically-determined or optional arguments that the e-

signature program can then locate and read. Creating auxfields for all arguments is another valid approach, but this results in a negligible difference in programming effort for the DLL programmer (a big conditional statement instead of many methods) and a significant increase in programming effort for the Blaise programmer (declaring and setting extra auxfields). Likewise, the e-signature result (whether or not the e-signature is successfully recorded or requires edits) is also returned directly to the DEP, eliminating the requirements for user input from the interviewer and resolving the presence of an image file via Manipula.

The alien can return values from our process back to the active record in DEP, and we can use the result to determine if a form was saved and the image file exists in the secure location. We eliminate the step of resolving discrepancies due to user error related to an interviewer misreporting a result via data entry as shown in the following flowchart (see Figure 7).

Figure 7. Processing Electronic Signatures with Alien Router



7. External Applications in Blaise 5

For the administration of an electronic signature application via Blaise 5, it will be necessary to first include the .NET application as a Windows Service. In the Blaise datamodel code, we can create a procedure that calls the service as an ALIEN.

As per Blaise 5 Help, in the .NET code for the service we define a ServiceHost instance and service endpoints for data exchange.

In our .NET procedure for the e-signature form, we can validate the parameters passed to the form as well as any activity within the form, and then export our relevant result (saved, did not save) as a WCF (Windows Communication Foundation) “int” service type to the active record. In addition, we could allow editing of the form to pass back a WCF “string” for more complex form results.

8. Conclusions

The use of the electronic signature form as a means of obtaining, documenting and saving crucial consents data has proven extremely effective in our household survey. The process mitigates user error; allows our home office processing staff quick access to the electronic documentation; and has proven to be effective and reliable in a complex data collection. There has been a significant reduction in paper management with the security risks and other problems associated with paper data collections.

As a platform, Blaise 4.8 has provided Westat with several efficient approaches to coordinate parameterized input data from the DEP with the extended capabilities allowed via .NET applications and processing. As we transform our processing to utilize Blaise 5 capabilities, we feel that enhanced platform will further advance our efficiency in processing this key data.