Generic System Interface Instrument

Curtis Ziesing, U.S. Census Bureau, Author/Presenter

1. Background

When the U.S. Census Bureau's legacy Computer-Assisted Interviewing (CAI) case management systems were designed back in the early-to-mid 1990's, one of the most problematic areas was the interface between the systems and the instruments. For our early surveys, system developers programmed their systems concentrating on the desired functionality for the end-users. At the same time the survey instrument authors programmed their questionnaires, concentrating primarily on the subject matter part of the instrument. The two pieces didn't come together until late in the design process and interface issues had to be resolved at the last minute. This caused problems in trying to get surveys fielded on time. Because there was no instrument available for system programmers to test while their system was being developed, interface issues could only be addressed once a "working" instrument was made available to them. (A "working" instrument is defined as one that includes the system interface pieces.)

As more surveys were converted to automated questionnaire data collection, it became more and more apparent that there was a desperate need for standards. These standards did evolve over time

These included:

- Standard Record Typed Input files for loading control and previously collected dependent data for the sample cases.

- Standard Interface fields to share data between the instrument and case management systems

- Standard File naming conventions so the systems would know how to process the file based on its file name.

One thing that was still lacking was a way to test the instrument interface with any newly developed systems until an automated instrument was available. As the Census Bureau was in the process of creating the next generation of systems to manage CAI collections at the same time it was creating its first Blaise instruments, it was decided to build a skeleton (or bare bones) instrument that would address the interface issues and allow system developers to test their systems early in the process so that the majority of the interface issues would have been resolved by the time the actual instrument was ready.

Some of the system changes that were occurring were:

- Converting from the DOS to Windows 2000 environment in CAPI

- Converting CATI from DOS to Browser-based with a central database repository for all telephone centers

- Converting survey instrument software from CASES to Blaise

- Storing case-level data as a BLOB field in databases for case transfer and access
- Running remote shell (rsh) from UNIX to NT platform for processing Blaise data

This "skeleton" instrument was designed to address these changes and allow system developers to test their systems while in the early stages of development. As new functionality has been added to the systems, the skeleton instrument has been used to prototype these changes. This paper will describe how the skeleton instrument was designed and constructed and the functionality it addresses.

2. Overview

The skeleton instrument was developed to provide the Census Bureau's CATI and CAPI case management systems with a means to test their interfaces with an instrument before the actual instrument is ready for testing. A computer assisted telephone interview (CATI) requires a different set of tests to be performed than a computer assisted personal interview (CAPI).

The instrument itself is quite simple. It is a shell, or one might say "a skeleton," that collects data only for the variables that the case management systems care about, since its purpose is to test the interfaces with the CAI control systems. The more complex piece is the process that was developed to automatically create the instrument, its associated scripts, and an input file (or preload data) based on the requirements of the particular survey instrument. There are six basic steps to this process:

1. Run a Blaise instrument that prompts the creator of the skeleton instrument for test-specific requirements. The instrument gets the name, survey period, and record types specific to the survey.

2. Generate the skeleton instrument appropriately based on the answers to the prompts. All the record types and CAPI or CATI code are added to a standard source file called **Inst.bla** and then processed into the skeleton instrument to be used for the specific testing purposes.

3. Generate an input file appropriately based on the answers to the prompts. The input file, if needed, may have four people in a single household or it may contain twenty people in several families. Based on the number of records requested these input files will be duplicated until enough records are created for the test.

4. Generate Manipula scripts for the interfaces with the case management systems. Because there are CAPI or CATI surveys, the Manipula scripts are different to handle the different interfaces with the control systems. Each of the Manipula scripts begins with a known set of standard lines of code, and then some specific lines of code are added to create scripts specific for the testing being done.

5. Create Blaise databases using the input file. This Blaise database will contain enough blocks, fields, and questions to allow for testing the control system interfaces.

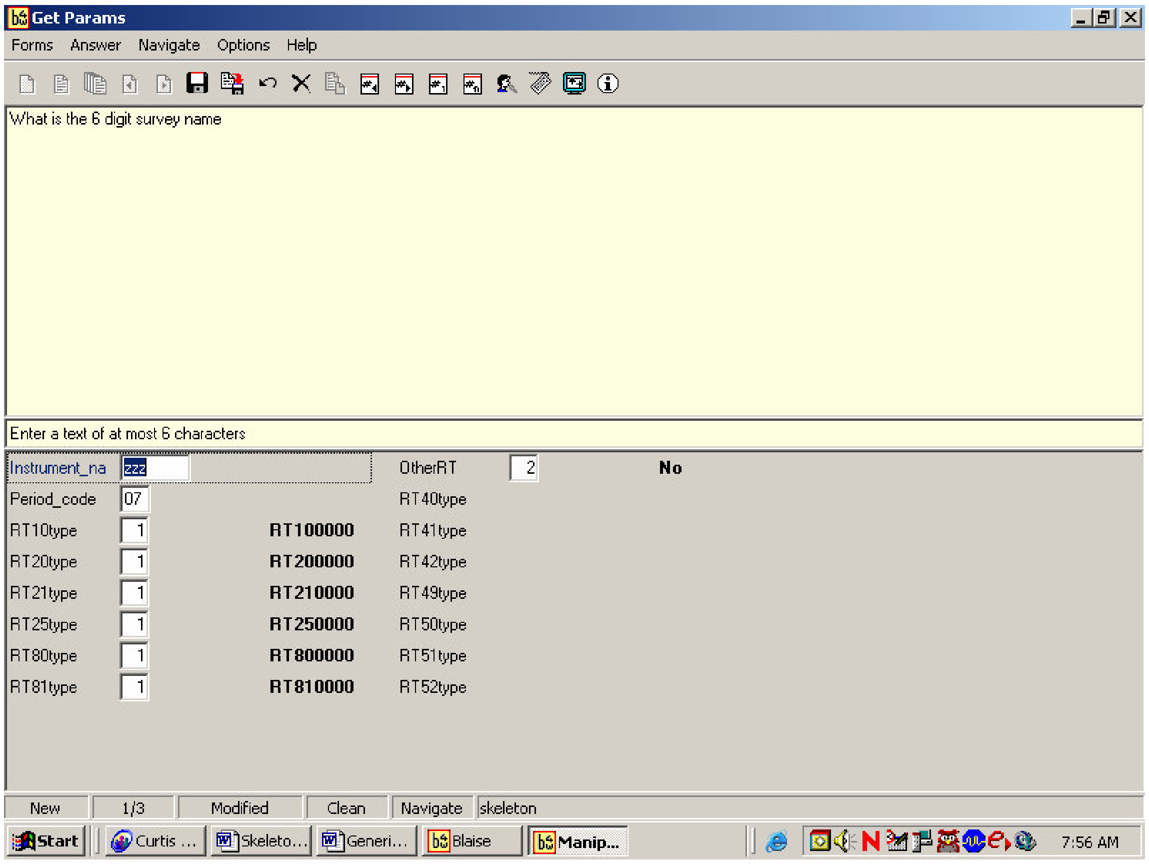
6. Package the instrument for delivery to the appropriate case management system.

3. Collect Information

The process begins with the user running the **"getinfo"** Manipula script, which in turn runs a Blaise questionnaire named **"getsurveyinfo"** that collects answers to a series of questions to determine how the skeleton instrument and associated files are to be created, what the testing environment will be (CATI or CAPI), and how the input file is to be created. Although the same types of scripts are used for CATI or CAPI, the structure of each script is considerably different. The skeleton instrument will create the scripts needed based on the answers to the questions.

3.1 Instrument Name and Interview Period

First the study name (which is unique for each survey collection period) is captured. Figure 1 shows a screen where the name of the instrument is collected.


Get Params
Forms Answer Navigate Options Help
New 1/3 Modified Clean Navigate skeleton
Start Curtis ... Skeleto... Generi... Blaise Manip...
7:56 AM

What is the 6 digit survey name

Enter a text of at most 6 characters

Instrument_na	zzz		OtherRT	2	No
Period_code	07		RT40type		
RT10type	1	RT100000	RT41type		
RT20type	1	RT200000	RT42type		
RT21type	1	RT210000	RT49type		
RT25type	1	RT250000	RT50type		
RT80type	1	RT800000	RT51type		
RT81type	1	RT810000	RT52type		

Figure 1. Collecting the name and record types to use

The case management systems use this information to create the appropriate directory structure when this instrument is installed. The skeleton instrument process also uses it to create the appropriate directory structure and to name the packaged file for delivery to the designated system. The answers to these questions will determine the files structure of the specific survey instrument. In Census Bureau systems, the study name consists of a 3-character survey acronym, a 1-character field that identifies whether it a production or re-interview survey, a 2-character panel code; and a 2-character interview period code. For example, a study name like acs_ag09 would indicate that this is for the ACS (American Community Survey), the "_" indicates it is an original data collection instrument for production, the “ag” is a unique panel code assigned to the survey (may indicate the calendar year of the survey), and the 09 would indicate the month of the survey. The Census Bureau convention also refers to the last 4 characters of the survey name, which is made up of the panel code and the survey period, as the survey_id and it is used as an integral part of the file naming convention for the various files used by the systems.

3.2 Record Types

The next step is to identify the various record types that are required for the survey’s input file. The Census Bureau uses a standardized set of record layouts for the various types of data that are typically included in an instrument as preload data. Each unique record layout is assigned a “record type,” and the input file for a particular instrument is made up of a set of these record types for each case. The record types fall into three categories:

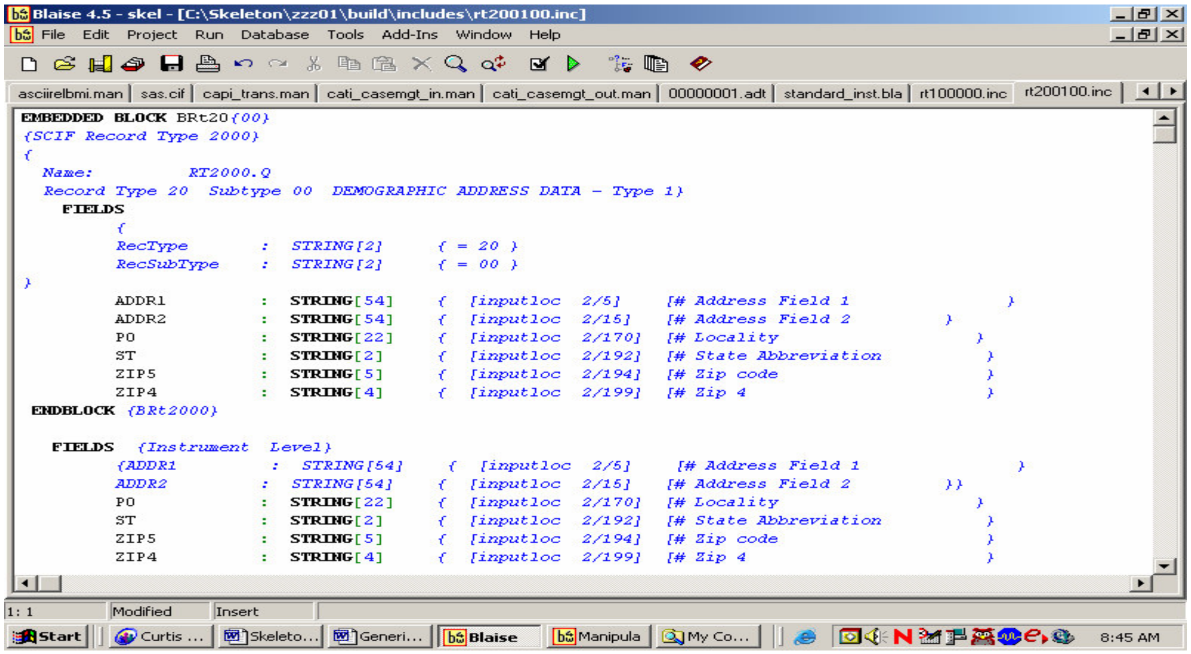
1. Those that contain data used only by the case management systems.

2. Those that contain data used only by the instrument.

3. Those that contain data used by both the case management systems and the instrument.

The skeleton instrument process is only concerned with record types that contain instrument data (categories 2 and 3).

Besides defining what record types are needed on the input file, this step is also defining some of the blocks that are needed in the skeleton instrument. Because the Census Bureau has pre-defined record layouts for input, we also have standard record type “include” files that define the standard variables on those record types. For each record type used by an instrument, that instrument will have a corresponding block of data. For example, if the record type for the address is 200100, the instrument will include an Rt200100.inc file, which will define the variables in a block named BRt20. By prompting for the record types in the getinfo script, the skeleton instrument process is then able to know which standard include files to put into the instrument. An example of a standard record type is provided in figure 2.


Blaise 4.5 - skel - [C:\Skeleton\zzz01\build\includes\rt200100.inc]
File Edit Project Run Database Tools Add-Ins Window Help
asciirelbmi.man | sas.cif | capi_trans.man | cati_casemgt_in.man | cati_casemgt_out.man | 00000001.adt | standard_inst.bla | rt100000.inc | rt200100.inc
1: 1 Modified Insert
Start Curtis ... Skeleto... Generi... Blaise Manipula My Co... 8:45 AM

```
EMBEDDED BLOCK BRt20{00}
{SCIF Record Type 2000}
{
  Name:          RT2000.Q
  Record Type 20  Subtype 00  DEMOGRAPHIC ADDRESS DATA - Type 1}
    FIELDS
          {
          RecType        :  STRING[2]     { = 20 }
          RecSubType     :  STRING[2]     { = 00 }
}
          ADDR1          :  STRING[54]    {  [inputloc  2/5]    [# Address Field 1             }
          ADDR2          :  STRING[54]    {  [inputloc  2/15]   [# Address Field 2         }
          PO             :  STRING[22]    {  [inputloc  2/170]  [# Locality                  }
          ST             :  STRING[2]     {  [inputloc  2/192]  [# State Abbreviation         }
          ZIP5           :  STRING[5]     {  [inputloc  2/194]  [# Zip code                   }
          ZIP4           :  STRING[4]     {  [inputloc  2/199]  [# Zip 4                      }
 ENDBLOCK {BRt2000}

   FIELDS  {Instrument  Level}
          {ADDR1          :  STRING[54]    {  [inputloc  2/5]    [# Address Field 1             }
          ADDR2          :  STRING[54]    {  [inputloc  2/15]   [# Address Field 2         }}
          PO             :  STRING[22]    {  [inputloc  2/170]  [# Locality                  }
          ST             :  STRING[2]     {  [inputloc  2/192]  [# State Abbreviation         }
          ZIP5           :  STRING[5]     {  [inputloc  2/194]  [# Zip code                   }
          ZIP4           :  STRING[4]     {  [inputloc  2/199]  [# Zip 4                      }
```

Figure 2. Typical record type block

It shows the definition of a block with the group of fields defined, and then after the block is defined the fields are repeated. This process is to make sure any changes to the data are done outside of the BRT20 block definition. The data stored in this block will remain unchanged throughout the interview process.

3.3 Environment

The next set of questions helps establish the environment in which the instrument will be run. The Blaise version is collected because Census Bureau case management systems allow the use of different versions of Blaise for different instruments.

The next question asks whether the instrument is for production, systems testing, or training. The Census Bureau has included a field on one of the input file record types for this information, and it can be used in various ways by the instruments and the case management systems. Instruments have used this information to follow a path in a training case that differs from the path in a production case. For example, for training cases you might want a reference period to remain static while it is dynamic for a production case. This can be controlled through the use of the “survey type” field.

Next, the paths to Blaise and WinZip on the user’s computer are captured for later use. This is done so that any user on the network can create a skeleton instrument. The Census Bureau generally sets up a users system with Blaise and WinZip in the same place but this can be adjusted for individual uses. In a Windows environment the default Blaise can be one version and the skeleton instrument can be created in a completely different version. The Census Bureau also uses different versions of WinZip to accomplish different tasks. One version is used to run command line parameter execution and another is used to run batch processes. The reason for establishing environment parameters is to allow flexibility in the use of different versions of Blaise running in different graphical user interfaces with different directory structures.

3.4 Input File

The questionnaire then collects more information about the input file that is to be created. An input file contains a set of sample data for each case. For first contact cases the sample data may be all that the input file for a case contains. For subsequent contacts to a household, the input file may also contain information obtained from previous contacts. This may include people's names and data. First the user must specify whether an input file is needed or not. The user may opt out of creating an input file if one will be obtained from another source, although the skeleton instrument is normally used early enough in the development cycle before test input files have been created. If an input file is needed, then the user needs to specify the following things:

1. The number of cases to create in the input file and how many people in each case? When creating an input file, the user should consider the type of test the input would be used for. The skeleton instrument allows for a 4 person per household input or a 20 person per household input file. Since the size of the initial .bdb file for a case can be relatively small, for load tests where we want to simulate the collection of data and increase sizes of the .bdb file, by selecting the 20 person household, one can more closely approximate the transfer of completed data (this was used primarily in the testing of our CATI environments).

2. If the instrument is for CAPI testing, the number of Regional Office codes to include in the file and how many cases to create for each Regional Office (RO) code. The testing done for CAPI may require the use of two or more RO test databases to test for such things as respondents moving between regions.

3. The caseid number to start with. The Census Bureau uses an 8-character caseid number as the primary key for the Blaise databases. It is generally assigned sequentially starting with 0000001, but there might be reasons a different sequence is needed. If multiple input files are desired, this allows the user to select blocks of case IDs for each input to avoid duplication of case IDs.

4. Whether the resulting Blaise database files should be zipped or not. The case management systems maintain the Blaise data for each case separately, zipped and stored in a database BLOB field, but depending on what the skeleton instrument is being used for, the user may or may not want the files zipped. Some testing is only concerned with importing or exporting data into and out of the instrument. The full process of unzipping a BLOB field may not be ready for testing until other tests are complete.

5. The name of the output file. The output file that is generated from the manipula script is actually the input file used to setup the Blaise databases. If the skeleton instrument is used for a full systems test, the user can specify the name of the file that the systems are expecting for the test. An input file will have the data in ASCII format ready to be loaded into the Blaise database by the setup manipula script also generated by the skeleton instrument process and ready for testing.

3.5 Output Documentation

Information is then collected about what type of output documentation is required. The user may choose to receive data descriptions for ASCII output or SAS dataset definitions for ASCII relational output by running the appropriate Cameleon scripts.

3.6 Root Directory

Finally the user is prompted for the root directory for where the process is to build the skeleton instrument and related files. The root drive letter will be added to the beginning of directory paths. The Census Bureau establishes a set of directories where the instrument will be loaded and run from. This directory structure can be created on any drive letter within the network environment. This way the generic instrument can be created in a network environment that can be shared by several users.

4. Build the Files

After all the questions have been answered it is time to begin building the files. At the conclusion of the **”getsurveyinfo”** questionnaire control is returned to the **”getinfo”** script, which in turn calls the **”skeleton_build”** script. This script constructs all of the files and prepares them for delivery to the various systems. A directory structure that corresponds to the directory structure on our case management systems is created beneath the root directory indicated above. An additional sub-directory (named “build”) is also created. This is the directory where all the supporting files of the build process are copied or created. This is also the directory in which the input file will be built.

4.1 Create the Input File

If an input file is required, the script will create one using the information previously collected about which record types are required, how many cases to create (by RO, if CAPI), and the starting caseid. See Attachment A for the section of script that creates the input file. The starting input file for the skeleton instrument is a predefined set of people’s names and data that would normally be collected during an interview. Based on the number of cases to be generated, one predefined file would be used to have four-person households or another predefined file would be used having twenty-person households represented.

4.2 Create the Instrument

Now it’s time to create the instrument. The starting point is the record type include files discussed previously. It then pulls in other include files specific to the collection mode of the survey. The instrument, when executed by case management, allows the user to enter some demographic data as well as values for various control items (such as outcome code) that will be passed out to the case management system for it to act upon.

4.3 Create the Scripts

Although the specifics of the scripts may vary from survey to survey, there is a standard set of scripts that all our CATI and CAPI instruments use. The following is the basic processing flow for CATI and CAPI:

Step 1: (CAPI only) The CAPI Case Management system creates a **transfile**, which contains parameters used by the CAPI Transaction script.

Step 2: Case Management program (CATI or CAPI) creates a **<caseid>.in** file, which contains information collected or updated in the case management system that needs to be updated in the instrument data (or Blaise database).

Step 3: Case Management program runs a Transaction script (**cati_trans** for CATI or **capi_trans** for CAPI). The Transaction script controls the remaining instrument-related steps. See Attachment B for an example of a CAPI Transaction script. Different "transaction codes" cause the Transaction script to perform different steps. This allows the Transaction script to be the sole communication point between the case management systems and the instruments. See Attachment C for a list of the various transaction codes the Census Bureau use.

Step 4: The Transaction script runs a Casemgt_In script (**cati_casemgt_in** for CATI or **capi_casemgt_in** for CAPI). The Casemgt_In script updates the Blaise database with the data from the **<caseid>.in** file.

Step 5: The Transaction script calls the instrument, using information from the **transfile** for CAPI and using a standard run string for CATI.

Step 6: When the interviewer exits the instrument, the Transaction script runs a Casemgt_Out script (**cati_casemgt_out** for CATI and **capi_casemgt_out** for CAPI). The Casemgt_Out scripts creates a **<caseid>.out** file, along with some other output files, which contain information collected or updated in the instrument that needs to be updated in the case management system.

Step 7: Control passes from the Transaction script back to the case management system, which will then update its data using the information in the **<caseid>.out** and other output files and put the case away.

Thus, the standard set of scripts includes **<mode>_trans**, **<mode>_casemgt_in**, and **<mode>_casemgt_out,** where **<mode>** is for a CAPI or CATI instrument type. Each is different in the way the instrument is loaded for the interview and what data is passed into and out of the instrument.

In addition to the scripts used in the interviewing process, a **setup** script is also required. This script is run on the Census Bureau's Master Control System (MCS) instead of at the CATI or CAPI case management level. This is the script that creates the Blaise databases using the data in the input file. Each of these scripts is created using a predefined file, which is read one line at a time, and then the new instrument specific script is written out, one line at a time, using the correct record types.

The skeleton instrument process will create each of the appropriate scripts, based on whether the user indicated this was to be a CATI or a CAPI instrument. See Attachment D for the code that creates "*CAPI_CASEMGT_IN.MAN*". To create this script the skeleton build process reads a line of text from a generic file and writes a line of text to the instrument specific file, adding record type include statements where needed. This is where answering questions at the beginning of the process is now being acted upon. See Attachment E for the code that creates the Setup script. The setup script is a little more complicated but basically is uses a predefined generic setup script and then adds the instrument specific include statements for the record types. Additionally many new fields are added based on some newer record types. This is one of the great features of the skeleton instrument is it's adaptability. There are about nine of these instrument scripts that are created by the skeleton build process.

The process then compiles all the scripts. This process changes to the working directory based on the defined directory structure. Then each of the generated scripts is prepared using Blaise b4cpars.exe. This process will create the MSU files, which are the prepared versions of the source. Later these files may be added to the zip file for distribution.

4.4 Create Blaise Database

The process then runs the setup script against the input file to create the individual Blaise databases. The setup script uses the specific record types for the skeleton instrument. The setup script is similar to the one that will eventually be used to load data into the database during production. If the user requested that the Blaise files be zipped, the individual zip files are created at this point. This is where the path to the WinZip application is used.

4.5 Zip the Files for Distribution

The instrument and scripts are zipped and named appropriately for delivery to whichever system is using it for testing.

4.6 Create the Supporting Structure

In addition to the standard files that are generated, a number of batch files are created which are used to run the various scripts when required. These batch files simulate the areas of the process that may still be under construction. The following batch files are created:

1. Setcase.bat — This batch file will run the Setup script, which loads the input file into the Blaise database. This is the process where names and phone numbers and other supporting data are loaded for the interview.

2. RunCAPI.bat — This batch file simulates the Blaise scripts that CAPI case management executes to run the instrument and receive the data returned from the instrument. This is the process that actually runs the instrument for the interview. Data collected during the interview is then extracted from the Blaise database and prepared as an ASCII file to be loaded into the CAPI case management system.

3. RunCATI.bat — This batch file simulates the Blaise scripts that CATI case management executes to run the instrument and receive the data returned from the instrument. This is the process that actually runs the instrument for the interview. Data collected during the interview is then extracted from the Blaise database and prepared as an ASCII file to be loaded into the CATI case management interface.

4.7 Generate Output and Descriptions

Finally, the process will generate the data descriptions or the SAS definitions depending on the user’s selection for output documentation. Depending on how many cases were created this process will compile each of the parts of the SAS output relational files and generate a Blaise database structure. The process will then use Cameleon.exe to run the sas.cif script file to make the SAS dataset with specific data from each case.

5. Uses of the Skeleton Instrument

The following describe some of the ways the skeleton instrument has been used at the U.S. Census Bureau:

5.1 GUI Laptop Case Management Testing

This was the initial system that prompted the need for a skeleton instrument. We were converting from a DOS based (Clipper) Case Management system running CASES interviews, to a Windows 2000 environment running Blaise. We were trying to use the same type of data exchange files between case management and the instrument that we used for CASES. Instead of a consolidated data structure stored in folders on a laptop, we were changing to one set of data files per case zipped up and stored in a BLOB field in an Oracle database. More than just testing the interface at this point we were also coming up with a standard processing strategy for handling the Blaise surveys. Many of the transaction scripts from the skeleton instrument were used as the starting point for production survey scripts for the initial surveys.

5.2 WebCATI Development Testing

Since the WebCATI system was the first system to use a browser for control, we needed to test the handoff between the control system and the Blaise instrument. They also required a tagged output file to be fed from the Blaise instrument to the WebCATI control system. As with the laptop case management testing, the scripts were initially developed for the skeleton instrument and then refined and deployed with the real instruments.

The skeleton instrument was also used in the initial load testing of the system. For the load testing we were not using live interviewers conducting live interviews; instead we were simply timing how long it would take to extract cases from the BLOB fields in the database, launch the instrument, and put the case down again. We were able to create the input file with varying size cases and conduct the load test without having to wait for input files from another division or a finished survey instrument.

5.3 New Feature Testing (Telephone Questionnaire Assistance)

The WebCATI system also introduced a new dilemma for handling incoming calls for a survey that employed telephone questionnaire assistance as one of its collection methodologies. In all of our other surveys, the case was called up by some identifying field – case ID or control number and the data already existed in the Blaise database format. This new feature required that a case be generated on the fly (instead of being pulled out of a database) and the control information for that case was then obtained during the interview. The skeleton instrument was adapted to allow for this type of transaction, and the skeleton instrument scripts were modified to handle a “TQA” transaction. Once again, the changes were made and tested using the skeleton instrument, and then they were moved into the production instrument once they had been proven.

6. Conclusion

The skeleton instrument was developed to provide case management with an way to test interfaces with an instrument and to test process control before and after the survey instrument is loaded. Many times the amount of time to develop an automated survey can be shortened if several processes are developed at the same time. Having a general survey set of interfaces can provide control systems a way of testing their interfaces with an instrument long before the actual instrument is complete. This skeleton instrument can provide some very specific instrument characteristics for testing. Only those fields that need to be passed back and forth with the control systems for any type of instrument can be created. Adding to that a generic set of features that can be used to load test the systems process improves the quality of our surveys.

Acknowledgements

I would like to acknowledge Mike for writing the skeleton instrument. I would like to thank Tom, Annetta, Mike, Sue, and Karen for writing, editing, and providing valuable input into this 9th International BLAISE Users Conference presentation paper.

Michael Haas, U.S. Census Bureau, Author/Developer
Thomas Spaulding, U.S. Census Bureau, Author/Manager
Annetta DePompa, U.S. Census Bureau, Author/WebCATI
Susan Galeano, U.S. Census Bureau, Editor/Version management
Karen Bagwell, U.S. Census Bureau, Author/Manager

Attachment A – Create Input File

```
Process CreateInput

SETTINGS
  DESCRIPTION = 'Replicates a one case SCIF into many cases '

USES

DATAMODEL NewSCIF
 FIELDS
  I_RecType       :  STRING[2]     {=  10 }
  I_RecSubType    :  STRING[2]     {=  00 }
  I_ROSTERINFO_10 :  STRING[12]
  I_CASEID        :  STRING[8]
  I_CTRLNUM       :  STRING[24]
  Blank1          :  STRING[2]
  I_SITE          :  STRING[2]
  Blank2          :  STRING[86]
  I_SURVEY        :  STRING[4]
  Balance         :  STRING[859]
ENDMODEL

INPUTFILE
   infile : Newscif ('orig.inp', ASCII)

OUTPUTFILE
  OutFile : NEWSCIF ( 'new.inp', ASCII)

  SETTINGS
    OPEN = YES
    CONNECT = YES

AUXFIELDS
  I      : INTEGER  {Counter}
  N      : INTEGER
  SW     : INTEGER
  R      : String[2]
  D     : INTEGER
  Surv   : String[4]

MANIPULATE
 R := Parameter(2)   {Site code}
 N := Val(PARAMETER(1)) {Number of cases to be created}
 Surv := Parameter(3)  {Survey ID}
 SW   := Val(PARAMETER(4)) {Starting caseid}

IF R = '' THEN R := '00' ENDIF
 D := N + SW - 1
IF D < 10 THEN
   For I := SW to D DO
```

```
    Infile.OPEN
    Infile.READNEXT     {Reads the first record of the input file
}
    WHILE NOT (Infile.EOF) DO
          { display (str(N) +'= N '+ str(SW) +'= SW '+ str(D) +'=
D ',wait)}
        IF  Infile.I_RecType  =  '10'  THEN      {  Determine  record
type and load appropriate date }
           Outfile.I_CASEID := '00' + R +'000' + STR(I)
```

… {more code here}

```
           OutFile.WRITE
        ENDIF
      Infile.READNEXT
    ENDWHILE
   INFILE.CLOSE
   ENDDO
{ENDIF}
{  okay to here}
ELSEIF D < 100 THEN
                IF SW < 10  THEN
                       For I := SW to 9 DO
                            Infile.OPEN
```

… {more code here}

```
ELSEIF

N < 1000  THEN

 For I := 1 to 9 DO
   Infile.OPEN
   Infile.READNEXT     {Reads the first record of the input file
}

   WHILE NOT (Infile.EOF) DO

      IF Infile.I_RecType = '10' THEN    { Determine record type
and load appropriate date }
```

… {more code here}

```
        OutFile.WRITE
      ENDIF
   Infile.READNEXT
   ENDWHILE
   INFILE.CLOSE
 ENDDO
```

```
For I := 10 to 99 DO
   Infile.OPEN
   Infile.READNEXT      {Reads the first record of the input file
}

   WHILE NOT (Infile.EOF) DO

      IF Infile.I_RecType = '10' THEN     { Determine record type
and load appropriate date }
```

… {more code here}

```
    ENDIF
   Infile.READNEXT
   ENDWHILE
   INFILE.CLOSE
ENDDO
```

… {more code here}

```
ELSEIF

N < 10000  THEN

 For I := 1 to 9 DO
```

… {more code here}

```
 {uncomment out the next section to handle up to 1,000,000 cases}

{ ELSEIF


 N < 1000000  THEN
```

… {more code here}

```
{ENDIF }
```

Attachment B – Transaction Script

Capi_Trans.man

```
Process Run_Interview

SETTINGS
  DESCRIPTION = 'Transaction processing'

USES

 INPUTMETA 'inst'
 INPUTMETA2 'outcomes'

 DATAMODEL  TRANSACTION  {  Meta  file  (i.e.  bmi  file)  for  TRANSACTION
record }
                       { Also used for logging transactions}
   FIELDS
     CASEID         : String [8]
     BLANK1         : String [1]
     CONTROL_NUM    : String [24]
```
… {more code here}
```
     Time           : TIMETYPE     {time of transaction}

 ENDMODEL



 DATAMODEL ERRORFILE {Writes out ERROR messages to the file}

    FIELDS
      message     :  String [70]
      ERRORNUM    :  String[3]
```

… {more code here}

```
 ENDMODEL

UPDATEFILE   {The survey database}

  UpFile : INPUTMETA ('00000001', BLAISE)    {314}
   SETTINGS
    CONNECT = NO
    OPEN = NO {Gets opened in the manipulate section}

INPUTFILE    {The TRANSACTION file that case management creates}
```

… {more code here}

```
AUXFIELDS

  Verify_ID    : STRING[8]   {Used for comparison of case's CaseID and
TRANSACTION CaseID}
```

```
  Verify_CTRL  :  STRING[24]   {Used  for  comparison  of  case's  control
number and CONTROL_NUM trans rec}

  AUX_TRANS   : STRING
  AUX_ERRFLG  : STRING[3]
  AUX_DBINFO  : STRING[35]
  RO_FR_ID    : STRING[5]

  Reslt1             :  INTEGER    {Stores result of call to script to
update instrument}
```

… {more code here}

```
PROCEDURE RunDep    {Conduct the interview}
   PARAMETERS
        IMPORT
            THECASE,THEINSTRUMENT : STRING      {this is the caseID data
file used as input}
   INSTRUCTIONS
             Reslt2:=  edit  (THEINSTRUMENT  +  '  /Mskeleton.bwm  /f'  +
THECASE + ' /k' + THECASE + ' /g /x')
            IF Reslt2 <>0     THEN   { The DEP failed to execute }
               Prob.OPEN
               Prob.Reslt_no := '2'
               Prob.Reslt_val := Reslt2
               Prob.message := 'could not interview case'
               Prob.WRITE
            ELSE
               Flag9.OPEN(thecase +'.inst.flg')
               Flag9.write
            ENDIF
   ENDPROCEDURE
```

… {more code here}

```
MANIPULATE
   InTRANS.OPEN
   InTRANS.READNEXT    {Read the TRANSACTION file }

   WHILE NOT (InTrans.EOF) DO

   THECASE        := InTrans.caseid           {CASEID}
   THEINSTRUMENT   :=  InTrans.instrument        {Blaise  instrument  .bmi
file}
   UPINSTRUMENT     :=  InTrans.update_case         {Manipula  program  to
update instrument}
   UPCASEMGMT        :=  InTrans.update_casemgt     {Manipula  program  to
update case mgmt}
   refinst        := 'outcomes'
```

… {more code here}

```
    ELSEIF InTRANS.TRANS_CODE = '09' THEN

          Caseidin(THECASE,UPINSTRUMENT)
          IF  Reslt1  =  0  THEN      {If  update  from  case  management  was
successful, run DEP}
          {Upfile.release}
{LoadCATI}          Runexternal





{{RegularCATI}                        Reslt2    :=    RUN('note_editor.exe
'+THECASE+',P,E,'+RO_FR_ID+' ',wait)}

{{RegularCATI}                   IF  Reslt2  =  0  THEN   {If  interview  was
successfully run, update case management}   }
                            Caseidout (THECASE,UPCASEMGMT)
{{RegularCATI}              ENDIF      }
          ENDIF   {reslt = 0}

      ELSE {unrecognised transaction code send message}
         AUX_ERRFLG := 'T02'
      ENDIF   {InTRANS.TRANS_CODE = '01' THEN }

      IF Intrans.log_trans = '1'  THEN
         Log.WRITE
      ENDIF
 ENDIF
{ display ('stop 1 ',wait)}
 IF SUBSTRING(AUX_ERRFLG,1,1) = 'T' THEN

    Prob.OPEN {added by Mike Haas}
    Prob.initrecord
    {format the TRANSACTION error message}
    AUX_TRANS  := InTrans.CASEID       +
                  InTrans.BLANK1        +
                  InTrans.CONTROL_NUM  +
                  InTrans.TRANS_CODE   +
                  InTrans.OUTCOME_CODE +
                  InTrans.AGENDUM_CODE +
                  InTrans.FLDREINT_FLG
```

… {more code here}

```
  InTRANS.READNEXT    {Read the TRANSACTION file }
ENDWHILE
```

Attachment C – List of Census Bureau CATI and CAPI Tranasction Codes

CAPI

01 – Post-interview action code update
02 – Late mail return - update action and outcome code
03 – SFR observed case - set FLDREINT from trans file
09 – Run the interview

WebCATI

17 – Case is a TQA case
18 – Run the interview - process <caseid>.in file
19 – Run the interview - no <caseid>.in file

Attachment D – Create CAPI_Casemgt_In Script

Capi_casemgt_in.man is a Manipula program for reading in the ASCII file CASEID.IN data for the instrument to use during the interview.

```
   Outfile4.OPEN(substring(start_dir,1,len(start_dir)) + '\' + study +
'\build\e-inst\capi_casemgt_in.man')
   Outfile4.INITRECORD
   WHILE NOT (Infile4.EOF) DO {capi_in}

         IF Infile4.datatext = ' INPUTMETA ''ske_le''      ' THEN
         Outfile4.datatext :=   ' INPUTMETA ''inst'''
         Outfile4.WRITE

         ELSEIF substring(Infile4.datatext,1,6) = '{RT 10'  THEN
            IF RT10 = '100000' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  100000}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif
            ENDIF
         ELSEIF substring(Infile4.datatext,1,6) = '{RT 20'  THEN
            IF RT20 = '200000' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  200000}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif
            ELSEIF RT20 = '200600' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  200600}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif

… {More types can be added here}

            ENDIF
         ELSEIF substring(Infile4.datatext,1,6) = '{RT 21'  THEN
            IF RT21 = '210000' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  210000}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif
            ENDIF
         ELSEIF substring(Infile4.datatext,1,6) = '{RT 80'  THEN
            IF RT80 = '800000' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  800000}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif
            ENDIF
```

```
       ELSE
       Outfile4.WRITE
       ENDIF
 Infile4.READNEXT
 ENDWHILE
outfile4.close
```

Attachment E – Create Setup Script

{create the setup script with the proper RT files}

```
   Outfile2.OPEN(substring(start_dir,1,len(start_dir)) + '\' + study +
'\build\e-inst\setup.man')
   Outfile2.INITRECORD
   WHILE NOT (Infile2.EOF) DO

      IF Infile2.datatext =   '    INCLUDE  "Rt100000in.inc"' THEN
         Outfile2.datatext := '    INCLUDE  "Rt' + RT10 +'in.inc"'
         Outfile2.WRITE
      ELSEIF   Infile2.datatext   =              ('                 IF
In1000.Rt1000in.I_RecType   +   In1000.Rt1000in.I_RecSubType   =   ''1000''
THEN')   THEN
         Outfile2.datatext :=    '       IF In1000.Rt1000in.I_RecType +
In1000.Rt1000in.I_RecSubType = ''' + substring(RT10,1,4) + ''' THEN'
         RT10vers :=substring(RT10,1,4)
         Outfile2.WRITE
      ELSEIF  Infile2.datatext  =       ('                OutFile.SEGMENT
:= In1000.Rt1000in.I_SEGMENT      {comment out for RT1080}') THEN
             IF RT10vers <> '1080'  THEN
             Outfile2.WRITE
             ENDIF
      ELSEIF     Infile2.datatext      =                             ('
OutFile.RT1000.SEGMENT                    :=  In1000.Rt1000in.I_SEGMENT
{comment out for RT1080}') THEN
             IF RT10vers <> '1080'  THEN
             Outfile2.WRITE
             ENDIF
      ELSEIF  Infile2.datatext  =       ('                 Outfile.LISTID
:= In1000.Rt1000in.I_LISTID       {comment out for RT1080}') THEN
             IF RT10vers <> '1080'  THEN
             Outfile2.WRITE
             ENDIF
      ELSEIF Infile2.datatext =      ('            Outfile.RT1000.LISTID
:= In1000.Rt1000in.I_LISTID       {comment out for RT1080}') THEN
             IF RT10vers <> '1080'  THEN
             Outfile2.WRITE
             ENDIF

      ELSEIF Infile2.datatext = '    INCLUDE  "Rt200000in.inc"' THEN
         Outfile2.datatext :=   '    INCLUDE  "Rt' + RT20 +'in.inc"'
         Outfile2.WRITE

… {More types can be added here}

        ELSEIF RT21 = '218000' THEN
           if substring(Infile2.datatext,1,11) = '{RT 218000}' THEN
            Outfile2.datatext :=   Infile2.datatext
            Outfile2.WRITE
           endif
        ENDIF
      ELSE
        Outfile2.WRITE
      ENDIF
```

```
 Infile2.READNEXT
 ENDWHILE
outfile2.close
```