Redesigning CAI systems to handle multiple authoring languages

Mike Haas, US Bureau of the Census

Background

After a period of evaluation, the Census Bureau made a decision to use the Blaise system to develop the questionnaire for the Consumer Expenditure Quarterly survey. Until that time, the primary data collection tool for most of our demographic **C**omputer **A**ssisted **P**ersonal **I**nterviewing (**CAPI**) and **C**omputer **A**ssisted **T**elephone **I**nterviewing (**CATI**) surveys was the **C**omputer-**A**ssisted **S**urvey **E**xecution **S**ystem (**CASES**) software developed by UC Berkeley. The decision to use Blaise made a significant impact on our CAI control systems. Not only were we in the process of converting our control systems at the data collection level from DOS based to Windows based, but we would also now be administering two different data collection software packages. A decision had to be made whether to build a new system for Blaise surveys or to re-engineer our current system to handle both authoring languages (and be able to accommodate other packages that might be desirable in the future). We decided on the latter approach.

The **C**onsumer **E**xpenditure **Q**uarterly (**CEQ**) survey was not the first experience the Census Bureau had using Blaise because the Failed Edit Follow Up of the **A**merican **C**ommunity **S**urvey (**ACS**) was authored in Blaise. A customized control system was constructed with the assistance of Mark Pierzchala of Westat for the collection of the ACS Failed Follow Up data. The CAPI and CATI portions of the ACS were collected with a CASES instrument and administered under our standard control systems.

The CEQ survey would be the first Blaise survey to be integrated into our standard collection and control systems and thus required the re-engineering of those systems. Once it was determined that we would now be administering more than one type of authoring language, we had to rethink the process.

Processing systems in the data collection process.

There are several systems that communicate with one another during the data collection process. This discussion will focus primarily on the process for demographic surveys as the majority of the field work is conducted for demographic surveys, although the process for Economic surveys is similar. The systems consist of

1. The **S**ample **C**ontrol **S**ystem (**SCS**)
2. The sponsor's survey specific processing system
3. The **M**aster **C**ontrol **S**ystem (**MCS**)
4. The **R**egional **O**ffice **S**urvey **CO**ntrol (**ROSCO**) system
5. The CAPI **F**ield **R**epresentatives **L**aptop **S**ystem (**FRLS**)
6. The CenCATI system.

See Figure 1 following.

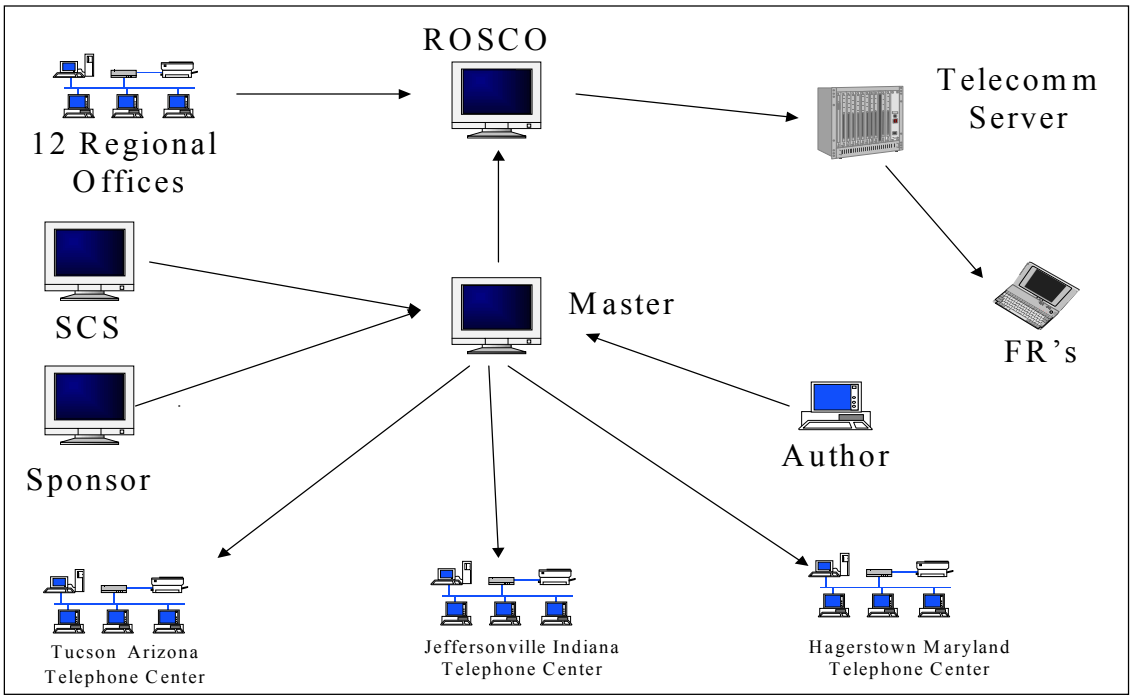

Figure 1: Tracking and Control Systems

A general synopsis of the flow follows:

The **S**ample **C**ontrol **S**ystem (**SCS**) maintains the universe of addresses initially obtained from the decennial census that is regularly enhanced by coverage improvement operations conducted throughout the decade. The SCS selects the sample of addresses to be interviewed for each of the major demographic surveys for each collection period and provides the control information for each case to be interviewed. The SCS delivers the file containing the control information for the sample addresses to the in-house survey sponsor. The sponsor's system merges any dependent data collected from prior contacts with the sample unit and produces a **S**ample **C**ontrol **I**nput **F**ile (**SCIF**). The SCIF is delivered to the **M**aster **C**ontrol **S**ystem (**MCS**). The MCS installs the sample cases and tracks their progress through the data collection operation. When completed cases are delivered back to the MCS, the MCS produces output of the collected data to the sponsor for post collection processing. These outputs either consist of ASCII data or proprietary data (Blaise or CASES) according to the sponsor's wishes.

The authors (Blaise and CASES instrument programmers) design the instruments according to survey sponsor's specifications. As the authors design the questionnaire, they also take into account the relationships between the instrument and the SCIF as well as between the instrument and the various control systems they share data with. Once the instrument has been programmed and tested, the author delivers the instrument to the MCS where it is merged with the SCIF, and the data is loaded into case level proprietary databases. The case level databases are stored in **B**inary **L**arge **OB**ject (**BLOB**) fields of an ORACLE database located on a Solaris/UNIX platform along with some case level administrative fields used to control and track the cases throughout the data collection process. Once the data has been loaded into the Oracle tables, the MCS distributes the CAPI cases to the ROSCO system and assigns the CATI cases to the appropriate Telephone center. The Regional Offices use the ROSCO system to manage and distribute the CAPI workload to the **F**ield **R**epresentatives (**FR**s). The FRs use a dialup connection to connect to the telecommunication server to pick up work and deliver completed cases. The

ROSCO system picks up the completed cases from the telecommunication server, updates status fields for the RO's use, and sends them back to the MCS to be output to the sponsor. Our current CATI system resides on Novell Servers and all of the cases to be completed at each telephone center are copied to the server to be worked on. Completed cases are sent back to the MCS each night to be output to the sponsor. In our new strategy for a revised CATI system, we would remove the necessity of the Novell servers and the telephone interviewers in the CATI centers will connect directly to the MCS Oracle database to pick up a case to interview. The call scheduling and case assessment programs will reside on the MCS. When the call attempt is completed, the case will be put back in MCS database for assessment and ultimately output to the sponsor.

Since our current production system has been evolving over the last 5 – 6 years, external users to the system (those providing inputs and receiving outputs) have finally become comfortable with the automated processes they have developed to interface with the system. We established standard file naming conventions that would categorize the various input and output file types so that one could easily distinguish the contents of the file by the name given to it. The file name identifies the mode of collection, the survey name and collection period, the month and day the file was created. The extension assigned to the file identified the type of contents contained within. Each survey also had its own directory structure on the Master Control System where the sponsor's data processors could deliver inputs and pick up outputs automatically from their systems. They were provided with programs in DOS, Unix and VAX (depending on their processing platform) that they would execute to deliver inputs from their machine to Master Control and pull outputs from Master Control to their platform. These programs would also do some housecleaning on Master Control. Now that the sponsors had a standard way of interfacing with Master Control, we wanted to make sure that their interaction with the new system as seamless as possible. Likewise all of our internal processes that handle the CASES instrument data have been fairly well standardized for all surveys. We also wanted to retain as much of the coding for those processes as well.

Standardized Interfaces

Since many diverse questionnaire instruments flow through the system, it was critical that the interfaces between the instruments and the systems as well as among the systems be as standardized as possible. We learned this early on in our administering of CASES surveys and we continued this practice as we migrated to Blaise. In order to expedite the processing between the systems from survey to survey, standard file formats were established to transfer control data between the systems. Each of these standard formats that we refer to as "record types" contain homogenous data that all surveys use. One record type contains all of the fields the Master control system needs to track a case, another contains address information, another contains information the ROSCO system needs to make interviewer assignments, etc. These record types are built into all instruments, although not all surveys use all fields.

In surveys authored with the CASES software, standard .q files (somewhat comparable to either .bla or .inc files in Blaise) are created for each record type and included in each instruments. Likewise in Blaise, we have created 2 standard .inc files for each record type. One that defines all of the fields to be defined in the instrument's .bmi file and one that defines the ASCII data model for that record type that will be used in the manipula script to load the Sample Control Input File (SCIF). In addition to the standard record type input .inc files, we create a control.inc file that defines all of the other fields needed by one or more of the control systems that are not already included on one of the standard input record types of the SCIF. These standard record types and control fields ensure that field names, sizes and types will be compatible with the fields in the control systems. When authors design a new instrument they begin with these standard record types and the control.inc files as a starting point. All of the fields defined in the record-type and control.inc files are at the datamodel level. The purpose for this is to expedite the data

exchanges between the control systems and the instrument. The section titled "Manipula script generation for building interfaces" will provide a little more detail on this process.

Differences in handling Blaise and CASES Data

One of the major differences between the approaches of accessing proprietary data between CASES and Blaise is that CASES used utility programs that work the same for all instruments. There is no direct relationship between the utility program and the instrument or its data until runtime. For example to load the dependent data into the CASES proprietary database, one used the **setup** executable. The authors handle the link between the dependent data file and the instrument fields during instrument design. When the instrument was compiled, an **INDEX** file was created that identified the relationship between each field and its corresponding storage location as well as where each field's data would be obtained from the dependent data file (just those fields that would receive input from the initial file). To load the data from the ASCII file into the CASES proprietary structure, one simply ran "**setup inputfile**" where inputfile contained the ASCII data to be processed. The **setup** program would then read the INDEX file and load the dependent data into the appropriate storage location. Another example is the process to produce ASCII data from the CASES proprietary data is accomplished by the CASES **output** program. By passing a file of field names and another file of caseids to the **output** program, **output** would generate an ASCII file containing the contents of those fields from the CASES data for the desired caseids. There was a fixed directory structure that CASES required to be in place for the software to function correctly, and the CASES utility programs acted on the instrument and the data that existed in the directory structure from which it was executed.

In Blaise, the same functionality is accomplished by preparing Manipula/Maniplus and Cameleon scripts. The programs that perform a similar process to what was done with CASES, are customized scripts that require the linking with the instrument data prior to runtime. These Manipula scripts need access to the compiled instrument during the prepare process. The instrument data model is highly integrated with the Manipula script and if there are structure changes made to the instrument, the manipula script that accesses it will most likely need to be re-prepared. Blaise allows a higher level of flexibility in accessing and formatting the output of ASCII data obtained from the proprietary data than CASES does, but also requires a little more customization. In CASES we could run the same program against all of our surveys without making any modifications, and therefore only delivered the program one time. In Blaise, since the Manipula scripts need to be compiled with the appropriate data model before they can be executed, survey specific versions are delivered to the client that will be executing the scripts along with the instrument. We did experiment with the **Get value** function of Blaise to see if we could pass in the name of the data model at runtime. This process worked but was very slow due to the size of the instrument and the number of fields being written out. It was much more efficient to compile the Manipula script for producing the output using the prepared instrument.

Another challenge we had to overcome was that most of our non-collection processing of CASES data was accomplished in UNIX. For Blaise surveys, we would try to accomplish these functions by rsh (remote shell) from our Solaris machine to an NT server and then run batch processes on the NT server to accomplish the tasks.

The above comparison was not to imply that the approach taken by Blaise or CASES was better than the other. It was simply to show the difference in the approach we would have to take into consideration to integrate the proprietary data with the control systems.

More Detailed overview of the instrument/system integration

After the survey sponsors provide the specifications to the instrument authors for the questionnaire and have determined the standard record types to be used for the specific survey in coordination with the Sample Control System (SCS), the SCS and the sponsors will merge the sample information with the dependent data from previous contacts and produce the Sample Control Input File (SCIF). The instrument authors pull the appropriate record type input .inc files for both the Blaise data model and the ASCII data models. They also pull the control.inc file. Then they program the instrument according to specs. In addition, the author writes a couple of Manipula scripts; one to load the SCIF data into the Blaise database and one that will be executed by laptop case management. The one executed by the laptop case management handles the interchange of data between case management and the instrument.

The author delivers the instrument and the manipula scripts to the Master Control System (MCS) and the SCS/Sponsor delivers the SCIF. The MCS pre-processes the SCIF by loading its database with some of the record types and updating some of the fields on the SCIF. One in particular is the assigning of unique case ID's to each case, which is used as the primary key for the case. The revised version of the SCIF is the one that will be processed by the manipula setup script. The MCS then runs a remote shell (rsh) to the NT machine containing the Blaise software. This shell executes the Manipula setup script to produce case level blaise databases. The name of the Blaise database files is the newly assigned case ID, (i.e. the primary key for the case). Each case's files are then zipped and stored into a BLOB field in the MCS database. The MCS determines the collection site where each case will be distributed and sets the site field accordingly.

For CAPI surveys, the ROSCO system reads the information from the MCS database and updates its databases accordingly. The ROSCO system will prepare the assignments for the Field Representatives (FRs) by pooling the appropriate Blaise data and the control information for the cases into a zip file, encrypting the file and placing it on the telecommunication server for interviewers to pick up. It also places the instrument and survey specific case management software on the telecommunication server for pick up by the FR's as well.

Included with the instrument are manipula scripts and a maniplus shell that handles the interface between the interviewer's case management system and the Blaise instrument. Since there are data fields that are updateable in both the instrument and case management system, the maniplus shell is a way to keep the common fields synchronized. The interviewer's case management system is a power builder application built around an Oracle Lite database. While the interviewer is viewing the cases in his/her assignment, a view of a number of the fields from the database is displayed to them. Some of these fields are updateable and some are not.

When the interviewer has selected the case they want to interview, they press the appropriate function key, and the case management software takes the blaise data for the case out of the BLOB field in the database and unzips it. It then creates a file called the "caseid".in file that contains case management fields that are to be updated in the Blaise data. The "caseid" is the primary key (e.g. 00000101.in). In addition to the "caseid".in file, a transaction file that instructs a maniplus shell (that is distributed with the instrument) what function to perform. The transaction file contains the primary key value for the case to be interviewed, the transaction code for the function that is to be performed, the name of the manipula script used to update the blaise data from the "caseid".in file, the name of the manipula script that will produce ASCII output ("caseid".out) from the Blaise data to the case management data at the conclusion of the interview. The layout of the "caseid".in file conforms to an ASCII datamodel that was prepared with the manipula program that updates the blaise data. Likewise, the layout of the "caseid".out file conforms to an ASCII datamodel that was prepared with the manipula program that updates the case management data.

The transaction code is used to determine which Blaise procedures to call in the maniplus shell. One transaction code is used to run the interview as outlined above, i.e. the sequence of updating blaise database - running interview – creating ASCII file to update case management. After case management processes the ASCII update file, it runs an assessor program against the control fields for the case to determine the next action to be taken. This action code is then passed back in a transaction file to update the blaise database through the same maniplus shell, but this time with a different transaction code. We also use the shell to indicate that a late mail return has been received for a case (for surveys where the CAPI work is following up on mail non-respondents), and also to indicate if a particular interview was observed by a supervisor so that the case would be screened out for potential re-interview. An illustration of this process is shown in figure 2 below:

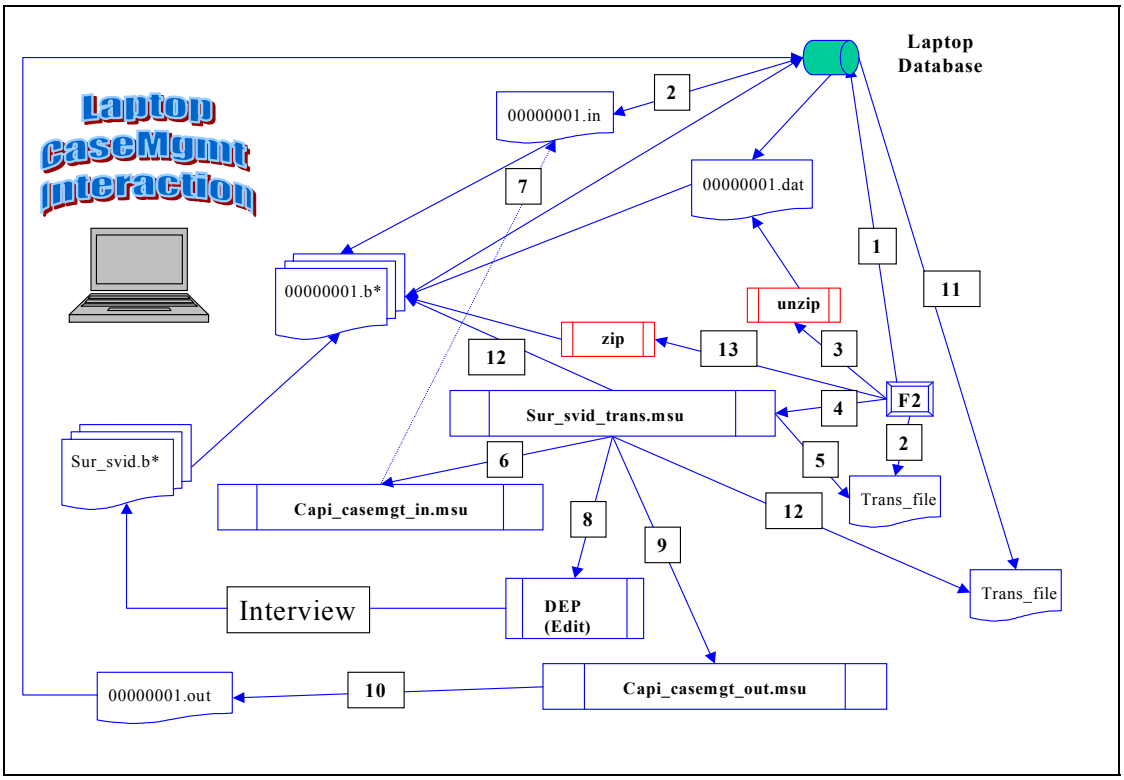

Figure 2: Laptop transaction processing

Steps in the process

1. The case management system writes the case's data from the BLOB field (zip file)
2. The case management system writes out the fields from the database that are to update the Blaise database for the case to a file named "caseID.in". It also writes out a transaction file (trans_file) that identifies the instrument meta file, the primary key for the case, the name of the Manipula script used to update the blaise data from case management, the name of the manipula script to update case management from the Blaise data, the transaction type (e.g. conduct an interview, update blaise data control fields…)
3. The case's zip file is unzipped into the appropriate Blaise files
4. The survey transaction maniplus shell is executed
5. The transaction script reads the transaction file to determine what function to perform.
6. For the interview transaction type, the following occurs: It executes the script to update the Blaise database.
7. The update script reads updates the Blaise database for the case by using the data in the caseID.in file.

8. The edit function in Manipula is then called to conduct the interview
9. After the interview completes, the script that writes out fields from the Blaise database to update the case management database is executed
10. This process produces an ASCII file named "caseID.out" which is used to update the case management system.
11. The case management determines what the next status of the case is and write out a new trans_file containing this new status (agendum)
12. The transaction script is again called, this time to update the agendum field in the Blaise data
13. Once all processes have finished for this attempt on the case, the blaise files are zipped back up and placed in the Blob field.

Some surveys allow the creation of extra units that are discovered at a sample address in the field before entering the instrument. The case management software knows which record types a particular survey uses and is able to construct an input file in the format of the original SCIF by pulling the appropriate data from the parent case. It then uses the same manipula script, which was used by MCS to load the original data, to load the Blaise database for this extra unit. It runs an ASCII to Blaise process.

Other surveys have coverage questions within the instrument that check for the existence of additional sample units sometimes referred to as "spin-offs or spawns". For example, our Survey of Income Program Participation (SIPP) interviews all the persons living in the household during the first contact. If on a subsequent contact, it is discovered that a persons living in the household at the time of the first contact has moved, the FR must attempt to locate the person that left the household. During the interview, the information for the whereabouts of the mover is obtained and those remaining in the household are interviewed as before. Once the interview has finished and the case put down, the same manipula script that writes out the control data for case management, will also create a new case for the mover. The new unit is created from the parent case with a Blaise to Blaise process.

Daily, interviewers will connect to the telecommunication server and the cases with a complete status and no further required on the laptop, will be pulled from the database and sent back. The ROSCO system will pickup the cases from the telecommunication server and place them in BLOB fields in the MCS database and update some of the control fields there as well as fields in the ROSCO database. The Regional Offices use the updated control fields in ROSCO to know the progress of their FR's work. MCS will then verify that the control data in the blaise database are consistent with the same fields in the MCS database (to verify that the BLOB field was updated correctly). For those cases where the 2 sets of control data are consistent, the MCS will produce output data for the cases that were delivered that day. It will produce either ASCII or ASCII Relational data for the completed cases, or if the data processors prefer, will create a consolidated daily file by concatenating the blaise databases for the completed cases into one. These MCS tasks are accomplished by manipula and or maniplus scripts that are run on the NT server by a remote shell (RSH) from the MCS UNIX box.

We are planning a similar approach for our CATI system except that the CATI clients in the telephone center will pull the data directly from the BLOB fields on the MCS system instead of having a database that resides locally. There will also be a CATI control table on the MCS that will contain the values of all of the control fields needed by CATI to determine the next disposition for a case. The first time a CATI workstation accesses a survey, the survey instrument will be pulled from the MCS database and unzipped on the CATI client. The instrument will remain there until the survey closes out. When the CATI interviewer requests either the next available case or a specific case, that case will be pulled from the MCS database and unzipped on the CATI client. It will then call a transaction processing maniplus script similar to the one used on the laptop, as described above. This script will read an ASCII file containing control system variables for the case that is written out by the CATI system. The script updates the corresponding fields in the blaise database files for the case and then opens the interview with the edit

function. At the conclusion of the interview, the script writes out an ASCII file of updated control information from the interview that is used to update CATI's master control file.

Manipula script generation for building interfaces

We have created generic maniplus scripts that through passing to them a set of parameters will generate manipula scripts for data interchange purposes. Some of the tasks these script perform are the following:

1. Generate ASCII output for specific fields (whose names were supplied in a file) for a specific set of cases (caseIDs also supplied in a file).
2. Update Blaise database fields from an ASCII file
3. Consolidate 2 or more blaise data base files into a single database file
4. Produce single case blaise database files from one consolidated file for all the cases contained in the consolidated file
5. Produce single case blaise database files for specific caseIDs (supplied in a file) contained in a consolidated file

Below is an example of the process for creating ASCII output for a list of control system variables:

A file containing the list of the fields to be included in the output is fed to the scripts as one of the parameters. A cameleon script, which is called from the maniplus script, will search the instrument's Meta files and attempt to locate in the blaise data model each field name passed to it. It then determines the attributes of the field and writes it to a .bla file. This .bla file will later be compiled to create an ASCII datamodel used for passing data from the instrument to the control system. In addition to constructing and preparing the ASCII datamodel for the output, for efficiency, this process also creates a filter file so that only those fields of the Blaise datamodel that are affected by the process need to be loaded into memory. So each time a Blaise instrument is opened, standard control data from the instrument is written to an ASCII file at the conclusion of the interview.

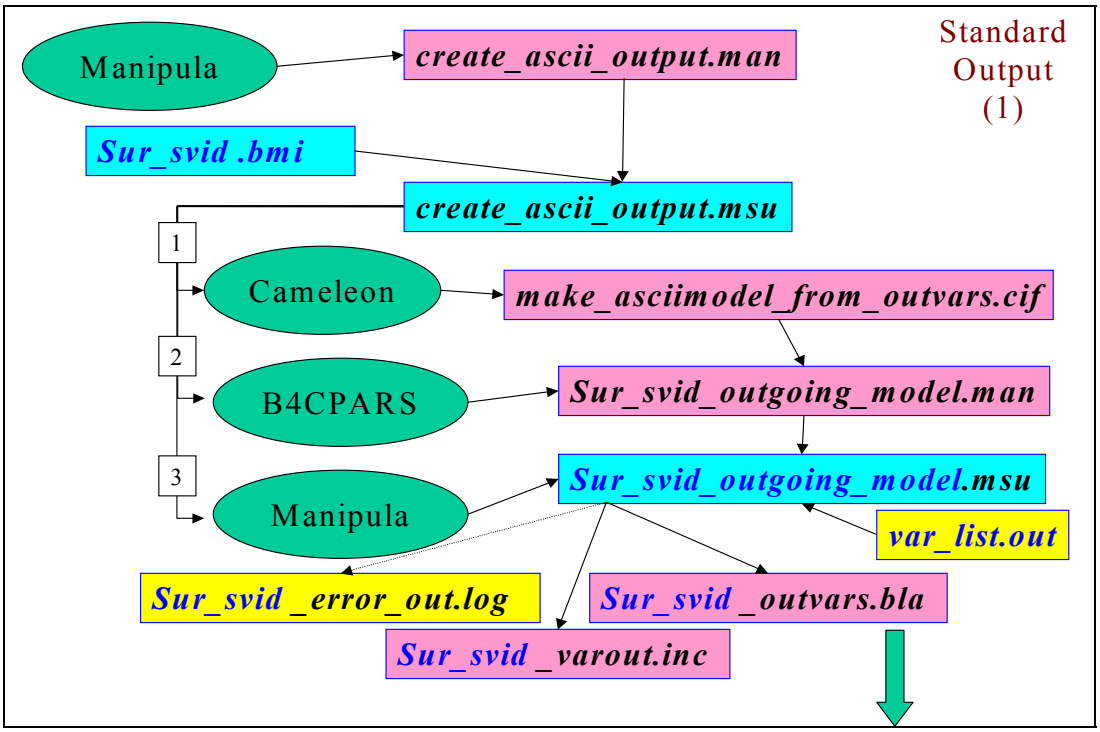

Figure 3: Creating output manipula script (part 1)

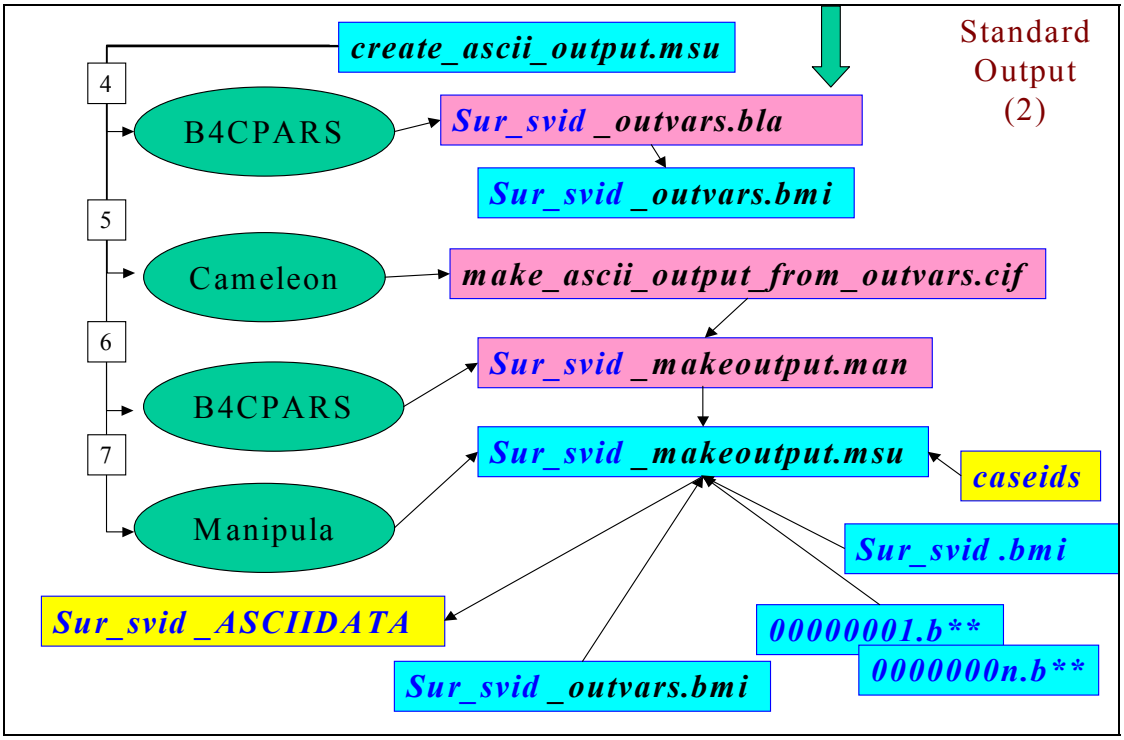

Figure 4: Creating output manipula script (part 2)

Steps in the process

1. The Maniplus shell that calls all of the subsequent procedures is written, prepared and executed
2. The Cameleon script that reads the meta definition for all the case level variables is created and executed to produce a manipula script that will ultimately compare a set a file of field names being passed to it against the meta data for the instrument.
3. The Manipula script created in step 2 gets prepared and executed to create a .bla file that defines the ASCII datamodel for the output file. It also creates an include file for that same set of variables, which when set as a filter expedites processing. It also writes to an error log file any field names included in the variable list file that are not found in the Blaise datamodel. In production, if there are any variables being asked for that don't exist in the blaise datamodel, processing stops until the process is rectified.
4. The .bla file created in step 3 is prepared and will be used in the last step to create the ASCII output file
5. The Cameleon script that creates the final manipula output script gets executed
6. The final Manipula script for producing output gets prepared
7. The last step produces the ASCII output by using the files created in the earlier steps. One of the parameters passed to the initial Maniplus shell determines whether or not this step gets executed.

The parameters passed to the maniplus shell are as follows:

Parameter (1) = .bmi file name
Parameter (2) = file of caseids
Parameter (3) = file containing variables to be output
Parameter (4) = location of blaise executables (drive:directory)
Parameter (5) = location of cameleon scripts (drive:directory)
Parameter (6) = working directory (drive:directory)
Parameter (7) = output filename
Parameter (8) = [X for no output] (whether or not to stop short of producing output)

Once the final manipula script has been created, it can be used to produce output on subsequent runs as indicated below.

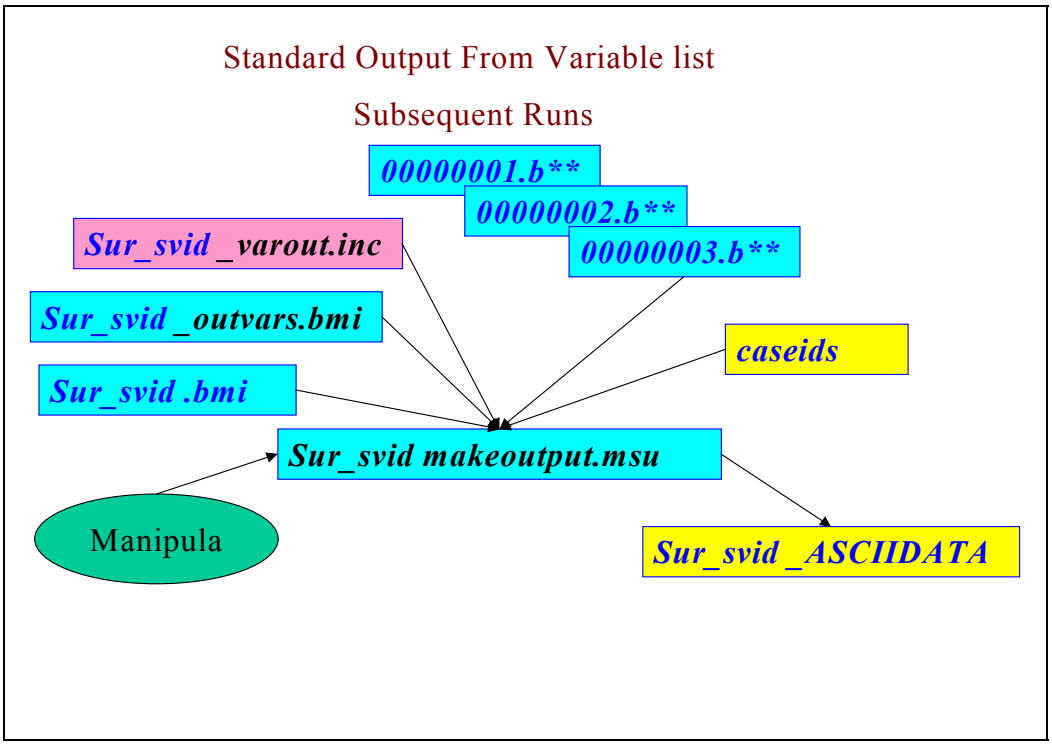

Figure 5: Subsequent Output runs

In summary, our goal is to have a standardize methodology in place to be able to field surveys in either CASES or Blaise as requested by the sponsor and to be in a position to accommodate other interviewing software as painlessly as possible should the need arise. If successful in our conversion, only those systems that will need to execute the data collection software will need to know which one to use. The Laptop case management, the CATI case management system, the Master Control and only those sponsors intending to use the proprietary data in their post-data collection processing, will need to know which software the instrument was authored in. The software for the SCS, ROSCO and those sponsor systems that are receiving only ASCII outputs should be unaffected by the choice of data collection software.