# Considerations in Maintaining a Production Blaise CATI System

**Kenneth P. Stulik, Total Service Solutions Inc. (for U.S. Census Bureau)**

## Introduction

The US Census Bureau's American Community Survey (ACS) is one of the largest continuous surveys in the world. The ACS is primarily a mail-out, mail-back survey which asks essentially the same questions as the Census Decennial Long Form. One part of the data collection effort is a Telephone Follow-Up (TFU) unit, which is essentially a CATI operation only for those mail returns that did not answer enough questions or had more people in the household than the form could accommodate.

The TFU CATI instrument is written in Blaise for Windows. What makes it different from most CATI instruments is that it must accept input data that represent the responses to questions in the mail questionnaire. This adds a layer of complexity since there are not only routines to maintain for periodic (daily, in this case) output of completed case data, but there are routines for the input of data as well. Control files exist to continuously assure that case counts between the various independent systems are synchronized. Reporting facilities gather information on daily caseload statistics, and automated mail routines notify a distribution list of either success or failure of the entire daily process.

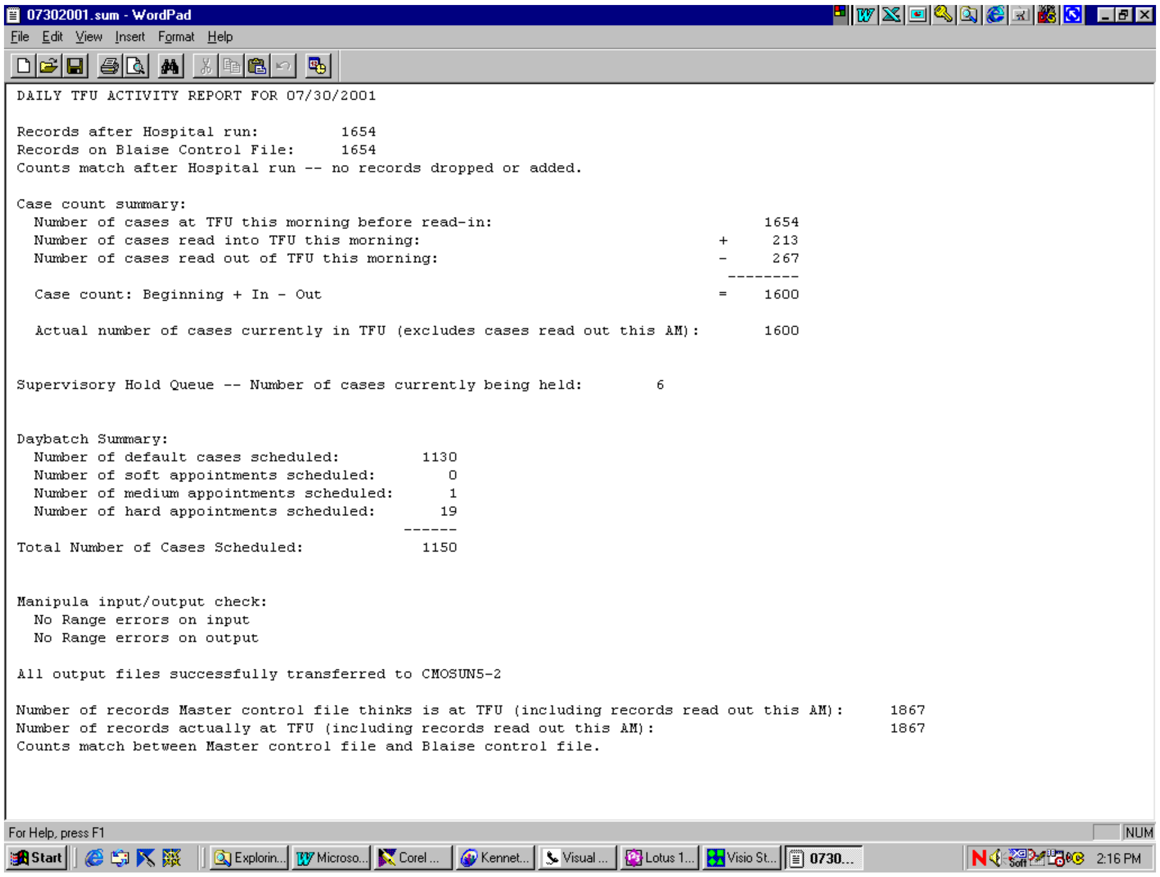
Integrating and maintaining all of these different aspects of the system can be very involved. Developing routine maintenance, upgrading instruments or Blaise software, addressing problems, and optimizing the entire process require much time and effort. Efficient resolution of all of these different topics is paramount when processing the 150,000 cases that currently come to TFU annually. This paper is intended to assist the survey administrator who is facing the upcoming challenge of having to prepare or maintain the survey effort for such an initiative, and the paper will address the various challenges and solutions for handling common issues.

## Operational Considerations

### Automated Routine Maintenance

Perhaps the most important function of a CATI Administrator is simply the routine maintenance of the production environment – making sure everything runs smoothly for the interviewers and supervisors so that they can do their job. A simple way of accomplishing this is to add some sort of automated e-mail notification to your data processing routines, so that any number of people can know easily if there are problems to contend with before opening for business. We use the UNIX mailx command to perform this task. These reports can be as simple or complex as desired.

The purpose of these summary reports is two-fold. They not only inform all key personnel of the status of the system on any given day, but they also provide basic case counts: number of cases read in, number of cases read out, number of cases in the daybatch, number of cases in the control file, status of transferred files, and whether things are in sync with other control files. With this type of system, it is easy to tell the general status of your entire CATI production system at a glance.



```
DAILY TFU ACTIVITY REPORT FOR 07/30/2001

Records after Hospital run:         1654
Records on Blaise Control File:     1654
Counts match after Hospital run -- no records dropped or added.

Case count summary:
  Number of cases at TFU this morning before read-in:                        1654
  Number of cases read into TFU this morning:                          +      213
  Number of cases read out of TFU this morning:                        -      267
                                                                         ---------
  Case count: Beginning + In - Out                                     =     1600

  Actual number of cases currently in TFU (excludes cases read out this AM):  1600


Supervisory Hold Queue -- Number of cases currently being held:        6


Daybatch Summary:
  Number of default cases scheduled:         1130
  Number of soft appointments scheduled:        0
  Number of medium appointments scheduled:      1
  Number of hard appointments scheduled:       19
                                           ------
Total Number of Cases Scheduled:             1150


Manipula input/output check:
  No Range errors on input
  No Range errors on output

All output files successfully transferred to CMOSUN5-2

Number of records Master control file thinks is at TFU (including records read out this AM):     1867
Number of records actually at TFU (including records read out this AM):                          1867
Counts match between Master control file and Blaise control file.
```

Figure 1. Screen shot of daily summary file.

Automated routines should perform a variety of functions beyond the read-in and read-out of data. Virtually all file archiving (covered in next section) can be handled in this way. Data integrity can and should be maintained in this process as well. The ACS automated routines perform a Blaise-to-Blaise Manipula transformation of the data and run Hospital daily. This aids in reducing Blaise database corruption, compresses Blaise database size, and optimizes primary & secondary keys. Reports of all kinds can be generated with these routines to keep track of any aspect of operational performance. Supplementary Manipula routines, daybatch creation, and file transfers also become part of this process. And detailed, dated logs of all of this activity make for a very robust application. Once fine-tuned and relatively error-free, this feature of the production system becomes the cornerstone of the CATI data processing operations and frees personnel up to do other work.

The ACS accomplishes this daily feat with a combination of tools. A WinAT task scheduler on a Windows NT Server launches a simple batch file every morning. The batch file in turn starts a SAS (third-party data management software suite complete with macro language and statistical functions) program which performs some of its own internal processing as well as the requisite calls to the OS to invoke Hospital and several Manipula routines. Flat files are transferred to a UNIX server which later looks for the presence of the deposited files and, if found, distributes them to a mail list.

How you configure your automated data processing is not so much the issue, so long as it is done. The process can range from simple to complex, and it can involve many sub-processes or few. It is best to use tools that you are comfortable with, tools that are well-supported within your organization, rather than tools that follow a given standard.

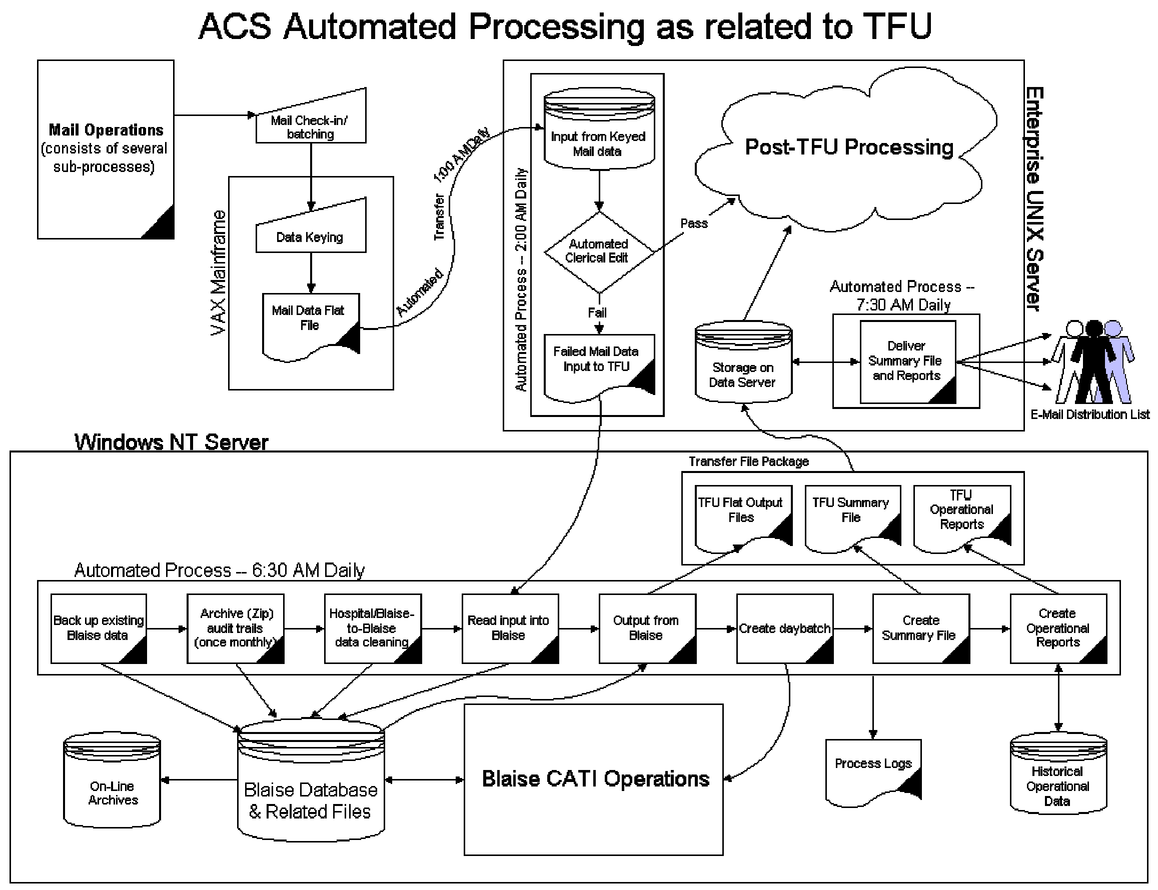
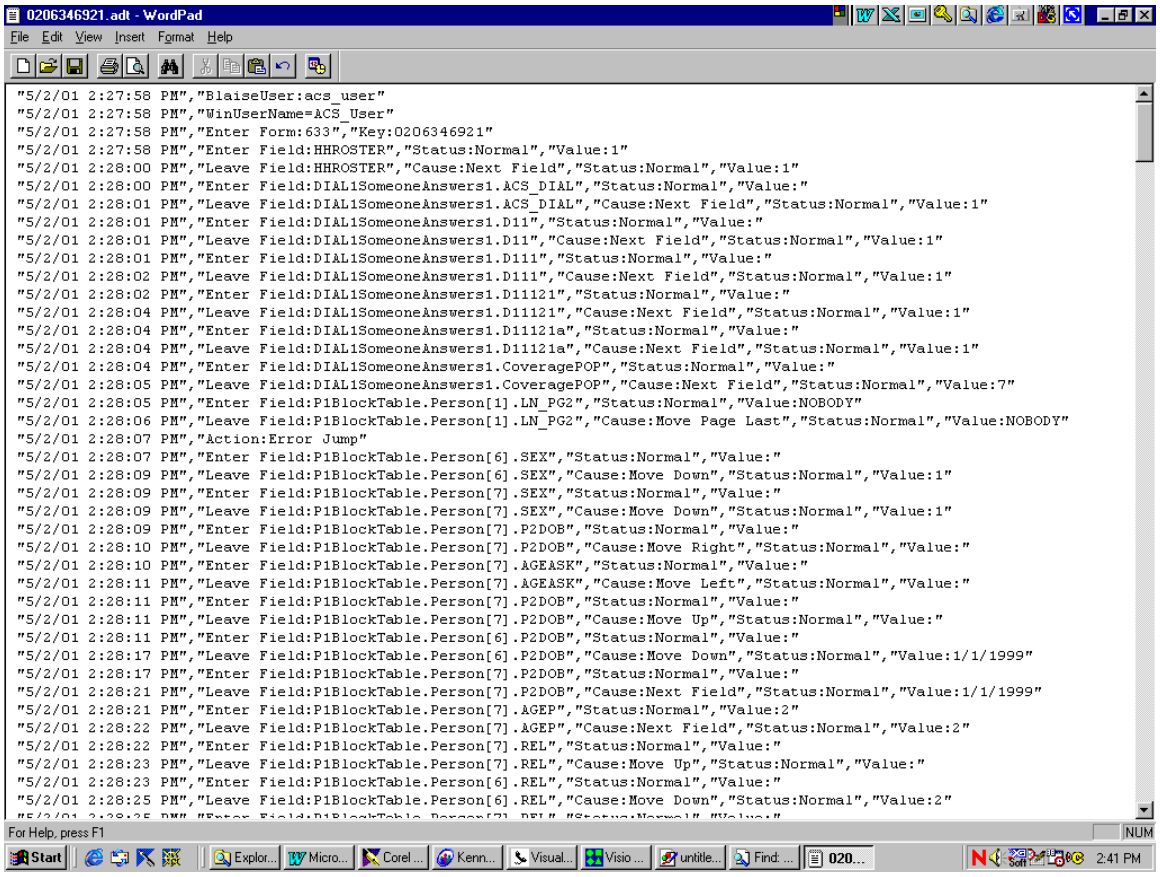

ACS Automated Processing as related to TFU
Mail Operations (consists of several sub-processes)
Mail Check-in/ batching
VAX Mainframe
Data Keying
Mail Data Flat File
Automated Transfer 1:00 AM Daily
Automated Process – 2:00 AM Daily
Input from Keyed Mail data
Automated Clerical Edit
Pass
Fail
Failed Mail Data Input to TFU
Post-TFU Processing
Enterprise UNIX Server
Automated Process – 7:30 AM Daily
Storage on Data Server
Deliver Summary File and Reports
E-Mail Distribution List
Windows NT Server
Transfer File Package
TFU Flat Output Files
TFU Summary File
TFU Operational Reports
Automated Process – 6:30 AM Daily
Back up existing Blaise data
Archive (Zip) audit trails (once monthly)
Hospital/Blaise-to-Blaise data cleaning
Read input into Blaise
Output from Blaise
Create daybatch
Create Summary File
Create Operational Reports
On-Line Archives
Blaise Database & Related Files
Blaise CATI Operations
Process Logs
Historical Operational Data

Figure 2: Data-flow diagram of automated processing operations for ACS with regard to TFU.

## File Maintenance

A natural side-effect of this type of automated production system is the continuous generation of many auxiliary files. One such file type is the audit trail. These useful files keep a text record of essentially every keystroke or navigational maneuver entered by an interviewer such that you can accurately determine what happened during any given interview. These files are extremely useful in troubleshooting problems with the Blaise Data Entry Program (DEP), since interviewers rarely recall the exact keystrokes they made in order to cause an error. The ACS TFU uses a modified audit trail dynamic link library which is designed to store audit trial info in text files named with the case's internal ID (also our primary key). This provides for easy access to a given audit trail, provided the case ID has been recorded or is able to be determined. But in the course of just one month, this can lead to more than 12,000 new files in your audit trail directory location! Most computers become very sluggish when asked to perform file operations on directories that contain a number of files of that magnitude, which can make the mere access of these files painstaking.



```
"5/2/01 2:27:58 PM","BlaiseUser:acs_user"
"5/2/01 2:27:58 PM","WinUserName=ACS_User"
"5/2/01 2:27:58 PM","Enter Form:633","Key:0206346921"
"5/2/01 2:27:58 PM","Enter Field:HHROSTER","Status:Normal","Value:1"
"5/2/01 2:28:00 PM","Leave Field:HHROSTER","Cause:Next Field","Status:Normal","Value:1"
"5/2/01 2:28:00 PM","Enter Field:DIAL1SomeoneAnswers1.ACS_DIAL","Status:Normal","Value:"
"5/2/01 2:28:01 PM","Leave Field:DIAL1SomeoneAnswers1.ACS_DIAL","Cause:Next Field","Status:Normal","Value:1"
"5/2/01 2:28:01 PM","Enter Field:DIAL1SomeoneAnswers1.D11","Status:Normal","Value:"
"5/2/01 2:28:01 PM","Leave Field:DIAL1SomeoneAnswers1.D11","Cause:Next Field","Status:Normal","Value:1"
"5/2/01 2:28:01 PM","Enter Field:DIAL1SomeoneAnswers1.D111","Status:Normal","Value:"
"5/2/01 2:28:02 PM","Leave Field:DIAL1SomeoneAnswers1.D111","Cause:Next Field","Status:Normal","Value:1"
"5/2/01 2:28:02 PM","Enter Field:DIAL1SomeoneAnswers1.D11121","Status:Normal","Value:"
"5/2/01 2:28:04 PM","Leave Field:DIAL1SomeoneAnswers1.D11121","Cause:Next Field","Status:Normal","Value:1"
"5/2/01 2:28:04 PM","Enter Field:DIAL1SomeoneAnswers1.D11121a","Status:Normal","Value:"
"5/2/01 2:28:04 PM","Leave Field:DIAL1SomeoneAnswers1.D11121a","Cause:Next Field","Status:Normal","Value:1"
"5/2/01 2:28:04 PM","Enter Field:DIAL1SomeoneAnswers1.CoveragePOP","Status:Normal","Value:"
"5/2/01 2:28:05 PM","Leave Field:DIAL1SomeoneAnswers1.CoveragePOP","Cause:Next Field","Status:Normal","Value:7"
"5/2/01 2:28:05 PM","Enter Field:P1BlockTable.Person[1].LN_PG2","Status:Normal","Value:NOBODY"
"5/2/01 2:28:06 PM","Leave Field:P1BlockTable.Person[1].LN_PG2","Cause:Move Page Last","Status:Normal","Value:NOBODY"
"5/2/01 2:28:07 PM","Action:Error Jump"
"5/2/01 2:28:07 PM","Enter Field:P1BlockTable.Person[6].SEX","Status:Normal","Value:"
"5/2/01 2:28:09 PM","Leave Field:P1BlockTable.Person[6].SEX","Cause:Move Down","Status:Normal","Value:1"
"5/2/01 2:28:09 PM","Enter Field:P1BlockTable.Person[7].SEX","Status:Normal","Value:"
"5/2/01 2:28:09 PM","Leave Field:P1BlockTable.Person[7].SEX","Cause:Move Down","Status:Normal","Value:1"
"5/2/01 2:28:09 PM","Enter Field:P1BlockTable.Person[7].P2DOB","Status:Normal","Value:"
"5/2/01 2:28:10 PM","Leave Field:P1BlockTable.Person[7].P2DOB","Cause:Move Right","Status:Normal","Value:"
"5/2/01 2:28:10 PM","Enter Field:P1BlockTable.Person[7].AGEASK","Status:Normal","Value:"
"5/2/01 2:28:11 PM","Leave Field:P1BlockTable.Person[7].AGEASK","Cause:Move Left","Status:Normal","Value:"
"5/2/01 2:28:11 PM","Enter Field:P1BlockTable.Person[7].P2DOB","Status:Normal","Value:"
"5/2/01 2:28:11 PM","Leave Field:P1BlockTable.Person[7].P2DOB","Cause:Move Up","Status:Normal","Value:"
"5/2/01 2:28:11 PM","Enter Field:P1BlockTable.Person[6].P2DOB","Status:Normal","Value:"
"5/2/01 2:28:17 PM","Leave Field:P1BlockTable.Person[6].P2DOB","Cause:Move Down","Status:Normal","Value:1/1/1999"
"5/2/01 2:28:17 PM","Enter Field:P1BlockTable.Person[7].P2DOB","Status:Normal","Value:"
"5/2/01 2:28:21 PM","Leave Field:P1BlockTable.Person[7].P2DOB","Cause:Next Field","Status:Normal","Value:1/1/1999"
"5/2/01 2:28:21 PM","Enter Field:P1BlockTable.Person[7].AGEP","Status:Normal","Value:2"
"5/2/01 2:28:22 PM","Leave Field:P1BlockTable.Person[7].AGEP","Cause:Next Field","Status:Normal","Value:2"
"5/2/01 2:28:22 PM","Enter Field:P1BlockTable.Person[7].REL","Status:Normal","Value:"
"5/2/01 2:28:23 PM","Leave Field:P1BlockTable.Person[7].REL","Cause:Move Up","Status:Normal","Value:"
"5/2/01 2:28:23 PM","Enter Field:P1BlockTable.Person[6].REL","Status:Normal","Value:"
"5/2/01 2:28:25 PM","Leave Field:P1BlockTable.Person[6].REL","Cause:Move Down","Status:Normal","Value:2"
"5/2/01 2:28:25 PM","Enter Field:P1BlockTable.Person[7].REL","Status:Normal","Value:"
```

Figure 3: Screen shot of sample audit trail.

After a case has been removed from the Blaise CATI system, the need for an audit trail file is minimized. Such files could be deleted from the audit trail directory location wiht some pre-determined frequency, such as once per week, once per month, or whenever the case is read out of the system. But we have found that these files can come in handy even long after a case has been processed in the CATI system. Since they are simply text files, with detailed information as to the exact field and value that has been accessed, they can be used to reveal (or even play back) any given series of keystrokes, or even aggregated and scanned for various values in specific fields. This can lead to a crude but effective method for data recovery.

Archiving these files is a sensible and straightforward task to do, and it can be easily automated. Since they are text files, they compress very nicely (usually more then 90%) using a utility such as PKZip, or any compression software using the LZ adaptive dictionary-based algorithm. This compression can be automated to run at whatever time period you determine to be optimal to your production processes. Then you can store large numbers of audit trails in archive locations that take up only one file and are less than 10% of the total size of the archived files, and you can easily uncompress and access them later if needed.

Another type of file that can be archived in this type of production system is the flat input or output file. The ACS TFU receives input files and creates flat output files on a daily basis. Saving these files in archive directories is not only prudent, but easy to manage since there are only a few files per day. Although they can take up several hundred kilobytes or even a few megabytes each, they can be stored without compression and archived manually once every year or so.

Blaise data files themselves are a third type of file that must be managed regularly. In the ACS TFU production environment, these files are not saved permanently nor are they archived on hard drive for a long time. We will save three business days worth of Blaise data files, but after the third day, the data files are deleted. They can at that point only be recovered by tape archives, which go back two months. The reason for this is that not only are the Blaise data files, which include Blaise database, primary key, secondary key, join key, telephone history, log, and metadata files (to name the more sizable types) prohibitively large to keep stored on disk, sometimes reaching 100 megabytes or more in aggregate size, but the need for them is very low in a production environment. That isn't to say that there is no use having access to any given case in the Blaise database for any given day. It's just that if you have to reconstruct something from more than three business days prior, something we have found to be rare, that it is usually possible to do so by reloading flat files or recreating results by using audit trail information than it is to pluck out certain cases from a previous database and insert it into a current file.

Other files that you may choose to manage might be those file types that are ancillary to daily read-in/read-out operations such as message (*.msg) files from Manipula processing or any external files that may be necessary to provide for a historical reporting facility (see section on Reporting Tools). Again, these files are small in size (less than 100K each), and they number only a few per day, so keeping them stored un-archived in subdirectories is not a file or disk management problem. Also they can be useful for historical reference or troubleshooting.

## Upgrades

No production system can be sustained for a lengthy period of time without upgrading some component. The Blaise system needs version upgrades because new desired features become available, or upgrades of new builds within versions are needed because of various software bugs. CATI instruments need upgrades because they also can be programmed with new features or to tweak that not-quite-correct procedure or include block. Automated data processing systems that supplement the Blaise CATI system need occasional bug fixes and enhancements. And the computer operating systems governing these various components occasionally need service pack updates or perhaps full migrations (e.g., Windows NT → Windows2000). I will address some of the issues involved every time an upgrade of any of these components occurs.

### Blaise system upgrades

Upgrades to the Blaise system are occasionally necessary for new features or for bug fixes. Statistics Netherlands has been very busy with Blaise for Windows 4.x, offering no fewer than 6 distinct versions with the 4.x architecture and dozens of builds among all of the versions. Sometimes simply choosing which version to upgrade to can be an unclear choice. Consult with the experts at Westat or at Statistics Netherlands if you are not certain of your upgrade path.

Normally, you upgrade from your existing version to a higher build number within the existing version, or from a given version to the next higher version in sequence, say from 4.1x to 4.2. Build upgrades with versions are usually performed because there have been bug fixes or processing efficiencies added to the previous build for known issues. Otherwise, you can upgrade from one full version to another for the purpose of acquiring new features that can make your overall processes more functional, robust or efficient. An example of this would be upgrading from version 4.1x to version 4.2 for the purpose of taking advantage of the many new CATI call-scheduler features.

Blaise version upgrades are usually not particularly difficult. When upgrading within a version level, for example from version 4.11 to 4.12, or even when upgrading between versions 4.0x and 4.1x or between versions 4.2x and above, the actual upgrade is relatively straightforward. Installing the new version is as simple as archiving or deleting your old Blaise version, creating a new directory, and installing the new version to that directory. Sometimes the same license files are applicable. If a registry update is necessary (this would have applied to earlier versions of Blaise 4.x), a separate utility exists to do that.

Occasionally, there can be intricacies to Blaise version upgrades. For example, upgrading from version 4.1x to version 4.2 or higher involved the OEM to ANSI conversion of all code, metatdata, database files, and so on. However, utilities and "wizards" for the transformation of all necessary Blaise files were provided, and with just some basic file management, making the upgrade and then transforming the files was not an overly complicated process. Future releases of Blaise versions will not likely involve a change of this nature, although other major changes could be involved. In any event, there will likely be a functional toolkit shipped with the product to facilitate whatever change is necessary.

The only real complication in performing a Blaise upgrade comes in managing multiple Blaise versions on your test environment. Almost always you will want to install the new version to test it, but keep the existing version around until the upgrade is complete. This has not been a problem for us, as we have found that multiple Blaise versions can exist harmoniously on the same server. When upgrading versions, we simply create a new directory and deposit the new version there. Any licensing information goes there as well, and if registry updates are needed we perform those as well. Then we point all of our shortcuts for Blaise programs in our test environment (such as DEP, Manipula, etc.) to the new version of Blaise, while keeping our copies of old production instruments and Blaise versions in their original places. Proper management of shortcuts and files allows for simultaneous testing of both old and new versions with little trouble.

The only caveat is that when upgrading to a new version that in turn updates the registry, we have found that all source code file become associated with the new version. So, for example, when invoking a current production piece of code, such as a .bla or .man file, you may expect the old version of Blaise to come up but instead the new version comes up but you do not notice. Then when you compile this piece of code, you may find that it does not work with your current production system. Careful attention to the particular version of Blaise under which you are compiling is prudent.

## Instrument upgrades

Upgrades to the CATI instrument tend to occur when a bug in the source code has been discovered and the code is modified to correct the bug. But instrument upgrades can also be used to create a custom feature that may not yet be intrinsically available in the Blaise software. Regardless of the reason for the deployment, careful testing should always be done. This paper will not detail all of the steps involved in testing an instrument upgrade, although as a general rule, when a change of this nature is made, you should thoroughly check the DEP instrument, any input and output routines, and any auxiliary functions that rely on the Blaise metadata.

There are two main types of instrument upgrades: those that involve a structure change and those that do not. Those that do not are much easier to deploy. Since there is no change in database structure, no Manipula routines need to be recompiled, nor is there a need for any data transformation. The upgrade can be as simple as overwriting the existing metadata files with the new ones.

However, those upgrades that do require a structure change are very involved and prone to problems for two reasons. First, such upgrades in a production environment require that the Blaise database be mapped into the new structure. Second, virtually every Manipula program that uses new metadata model must be recompiled. If this should involve new input or output variables, that complicates the process even further, since the Manipula routines that perform read-in or read-out must be modified to accommodate the new variables. This additionally requires verification that the input/output routines are running correctly and a thorough examination of the database afterwards. While the actual amount of additional labor to do this type of upgrade is not extensive, the high possibility for error requires that you spend a significant amount of time testing before, during and after the actual upgrade.

Frequently, multiple bug fixes will be deployed in a given upgrade because in the time that it takes to change and test the new code, other bugs may be discovered. This is especially the case when we identify a bug or an decide to add a new feature that requires a structure change. Since we extensively test our new instruments when making even one small alteration that requires a structure change, it is efficient for us to deploy many bug fixes at once. We may sometimes many go months between instrument upgrades when a structure change is involved. However, for upgrades that do not involve structure changes, we will usually deploy those more rapidly and without extensive testing.

## General considerations

We offer several considerations for testing an instrument upgrade, especially a major one in which a structure change is involved. First, it is highly advisable to have a separate testing environment, one which features a server of a similar type with similar operating system and having similar client stations as the actual production environment. For the ACS, we keep a complete, almost mirrored copy of the actual production environment on our test server. Under a slightly different root directory structure, there will be a test environment that has all of the new code to be deployed. In this area not only will subject matter testers give the new instrument a thorough examination with regard to the new structure, rules, CATI outcomes, textual content, and layout, but also all of the Manipula routines associated with the new metadata can be tested as well.

Second, just prior to conducting the upgrade, a dry run can be performed, all on the test server, in which the mock production environment on the test server is promoted to the new instrument and associated files. This not only serves as a test, but allows for practice as well. Furthermore, this methodology allows you to copy most compiled files from the test server to the production server when doing that actual upgrade, and this is usually much easier than recompiling everything at once.

Third, when upgrading, thorough and readily-accessible backups of all data, metadata, and source code files is essential. This allows for quick and easy recovery from any mishaps that may occur during the upgrade process. These backups may take up a considerable amount of space, but they need not be kept around indefinitely. As soon as the upgrade is complete and operational to the point that you are sure you will not need to recover from them, then these backup files can be deleted or archived to tape.

A fourth consideration when conducting a major upgrade is direct impact on interviewer down-time. Normally, you will not want to conduct an upgrade during normal business hours, as this will waste interviewing resources for an indefinite period of time. It is best to conduct major upgrades during off-hours. This gives you the flexibility to conduct the upgrade at a moderate pace, which should minimize the chance for errors, and it also allows for extra time if there are complications.

A fifth and final consideration when upgrading is the possibility of retrograding if necessary. If for any reason the upgrade cannot be carried out, say because of a significant bug that was not uncovered during testing or any other irreconcilable problem due to differences in the testing and production environments, you must be prepared to retrograde to the previous instrument or Blaise version. This is almost always undesirable, but it is better than having a system that is significantly deficient in some way. Retrograding is a very simple process if you have properly backed up all of the appropriate files.

## Reporting Tools

A successful Blaise CATI operation must have a method for evaluating its performance on many levels. Reporting tools can provide insight as to how well the interviewing unit is staffed. Perhaps there are too many interviewers, or too few. Perhaps the overall number is correct, but too many during some periods and not enough during other periods. Reporting tools can also track individual performance for purposes of monitoring new interviewers to see that their productivity is at acceptable levels.

Reporting tools can also uncover group response rates, dial outcomes or any other call-level related statistic, and these can be broken down by number of dial attempts, time zone, hour of day or whatever other operational variable that is needed. This can be useful for determining things such as whether or not you are making enough or perhaps too many dial attempts per case, whether or not you are calling during optimal hours, or whether you are properly staffed and configured to handle time zone differences.

Reporting systems can be devised in a number of ways. Certainly, use of BtHist, Manipula, Maniplus, and Cameleon allow for extensive management of Blaise data. Developing reporting facilities with these tools is limited only by the extent of your organization's Blaise programming skill. However, third-party tools can be used as well. The ACS uses a combination of Manipula and the SAS statistical analysis system to manage the Manipula output and produce several canned frequency tables every day, week and month. The unit supervisors use BtHist during the business day to track operational progress by the hour.

An ideal tool for use that is currently in development within the ACS is the dynamic web-enabled report. Such a tool will theoretically allow any user anywhere within the organization's firewall to immediately access any report for any time period or to conduct real-time queries on aggregate, historical corporate data. The drawback to this kind of system is that it is not easy to construct. It requires programmers and designers who are capable of designing HTML or Java-based web applications and integrating them with the Blaise system, as well as systems administrators capable of configuring web servers and broker facilities that allow such web applications to operate.

Whatever method is selected is not of major importance, so long as some kind of operational analysis is being done. Without it, there can be no reliable way of tracking operational efficiency and progress.

## Problem Resolution

One of the greatest challenges facing a survey administrator is handling problems that disrupt production activities. Some problems, such as power outages or crashed networks, are beyond the control of the administrator and must be tolerated. But most problems involve some kind of software issue, and when these problems arise, it is the responsibility of the administrator to effect a timely response.

The most threatening problems are those that bring the entire production system down indefinitely, or those that cause even a small amount of data loss. These problems require immediate action. There are several potential sources of errors that can bring production to a grinding halt: a Blaise system bug, a data corruption issue, an instrument bug, or even a network/OS/hardware problem.

One example of a crippling problem that involved data loss surfaced when we once did a major upgrade to our instrument. At the time, our read-in routines used a complicated ASCII-Relational method to process input data. Testing the read-in routines independently revealed no problems, but when we attempted to deploy in production, the input data did not correctly mesh with the existing Blaise database, resulting in loss of certain data fields upon input. When we tried to adjust the ASCII-Relational blocks upon read-in, that could not be successfully done. The solution was to immediately retrograde the instrument, redesign the read-in routines in a simplified ASCII method, test, and re-deploy.

Another example of a problem threatening production involves sections of data dropped by the Blaise system when BtMana failed to reassigned a case's future status. Initially, these problems seemed relatively harmless, simply causing a case to receive default instead of the correct call scheduler treatment. But we soon discovered that chunks of data were missing form a few cases. We were able to correlate these cases with missing data to cases that had missing FutureStatus in the call scheduler. With some data samples from ACS, Statistics Netherlands was quickly able to determine that this was caused by an archaic byte inherent to our data model improperly interacting with a newer version of BtMana. Statistics Netherlands was able to immediately prepare a new build of Blaise for us, we quickly tested, and deployed the new Blaise version. We were able to use some Manipula routines to re-populate the cases with the missing data.

Regardless of the type or nature of the bug, the first step is to troubleshoot it as thoroughly as possible. Generally, issues with data corruption or missing data are related to the Blaise system somehow. We have found over the years that normally the instrument, operating system, network, or server is not responsible for errors of this kind. Issues which involve performance problems are usually tied to the design of the instrument, the overall configuration of the files in the survey, the network or the OS. Issues involving the system freezing between CATI calls usually are attributable to the Blaise system.

Sometimes administrators are faced with problems that make interviewing difficult or inefficient for the interviewers but do not significantly stop productivity. It seems that the majority of problems fall into this category. These can be tricky to resolve because an administrator must balance end-user needs, the feasibility of the timing of the solution, and the threat to data integrity based on whatever solution is chosen. For example, we encountered a seemingly frightening error using version 4.12 in which one interviewer's workstation would lock up, then all other interviewers would lock up when they attempted to retrieve the next case from the call scheduler. At first, this error seemed very serious and we assumed we would have to make an immediate correction. But as we investigated, we realized that: 1) the problem was very intermittent for us – once every few months, 2) the problem was easily remedied by finding the first offending client workstation and rebooting that workstation, and 3) the problem appeared to cause no data loss. Eventually the recommended solution to this problem proved to be to upgrade to Blaise version 4.5 build 640. But since we were not prepared for that deployment, and since the total amount of interviewer down time was about 15 minutes every couple of months, we decided not to rush the issue and wait until our upgrade was ready. This is a classic example of a situation in which the instinctive urge to immediately correct a problem would not have been the best choice.

Other problems may require diligence and patience to solve. At one point, our interviewing crew of 24 people began experiencing DEP performance problems, in which during certain periods there would be delays of up to 20 seconds between questions due to the instrument response time. This of course was a major problem since a few respondents became very impatient and would abandon the interview. At first the problem was very difficult to pinpoint, because it coincided with other events on the Local Area Network (LAN) that would have been likely candidates to cause additional network traffic and produce wait states. But eventually these were found to be red herrings, as we discovered when using packet sniffers on the local router. Additional feedback from the interviewers helped us to pinpoint the exact times that the system slowed. Finally, we decided to conduct emulator testing on the CATI LAN. We found that when we ran a test on a separate but similar directory structure on the production server, we experienced similar delays in response time. But interviewers simultaneously in DEP in a different directory structure showed no slowdown. With the knowledge that the problem was instrument specific and occurred when a minimum threshold number of interviewers were active, we were then able to pinpoint the problem to our external lookup files. Placing those external files locally and reconfiguring the shortcuts proved to be the permanent solution.

When troubleshooting, use feedback from your interviewers, but be sure to verify and quantify things. Sometimes interviewers report certain conditions (e.g., "the computers are slow"), but don't use relative terms, use quantifiable terms (e.g. "The response time is 2 seconds"). Interviewers can also inadvertently exaggerate, since perceptions can become distorted when handling hundreds of similar cases per day, so be sure to verify claims using audit trails and logs where possible. Also, be sure that reported problems are in fact unique and not just a minor variation on another problem that is already being addressed.

Expert help is available from Westat and ultimately from Statistics Netherlands. Know when to call them in. But be sure to have exhausted all standard troubleshooting procedures first. With the ACS having been one of the first continuous survey efforts to use Blaise in a unique mail follow-up mode as opposed to a traditional CATI mode, there were a number of bugs we encountered with DEP, Manipula, and data corruption. Both Westat and Statistics Netherlands were pivotal in giving us direction as to how to correctly diagnose the problem and in developing workarounds or interim builds to address our problems. Without their assistance, our continued progress in this effort would have been virtually impossible.

There are several steps you can take to minimize the chance that your system will be subject to problems:

C Have the latest build of your Blaise version tested and in place in your production environment.
C Assure that your Blaise instrument is coded as efficiently as possible, minimizing excess code, minimizing file access, and having ample comments in place to assist troubleshooting.
C Make sure your server is optimized – run defragmentation regularly, scan for viruses regularly, have the latest operating system version (service pack) in place, run diagnostics intermittently.
C Run Blaise data optimizing tools such as Hospital or a Manipula Blaise-to-Blaise transformation regularly, even daily, to minimize chance of data corruption and optimize keys.
C Minimize clutter – archive or delete old logs, audit trails, Blaise databases, older metadata versions, or any other file that accumulates regularly and takes up significant storage space.

One other thing you can do to really minimize your problems is to set up your CATI environment on a separate server running within a separate LAN. We do not have the luxury of such a set-up, but the advantages in doing so are tremendous. One major advantage of such a configuration is that you can expect pretty consistent performance from your network. Another major advantage of such an isolated environment is that when troubleshooting, so long as you haven't made changes to the server or the LAN, you can usually be sure that any problems you may have are not attributable to them.

## Conclusions

When maintaining a continuous CATI survey operation, it is critical to have systems and procedures in place to handle both mundane and unusual situations as they should arise. An automated maintenance routine to process daily input and output data, generate reports, and alert personnel to problems is the centerpiece of any continuous survey effort. These features not only simplify day-to-day operational management for a small staff, but they allow optimization of the interviewing staff as well as data quality. Also, it is highly advisable to keep the complexity of such systems at a level that your organization is comfortable with. Having intricate systems and procedures for your survey administration may be appropriate, but they should also be within the scope of your ability to maintain and troubleshoot.

Established upgrade procedures, coupled with a well-structured test environment, are also a cornerstone of a successful survey effort. Without these, either the ability to upgrade software and instrumentation is compromised, or any upgrades performed run the risk of causing serious complications in the production environment. Thorough planning, testing, and documented procedures are a necessity for any administrator.

Quick and prudent problem resolution is another necessity. Most problems do not require immediate resolution, but all issues should be given an appropriate amount of attention under an overall plan to optimize performance of the unit – software, hardware, and personnel alike. Under such conditions, not only does data quality improve, but team morale increases as well.

By adhering to these practices, the overall functional quality of any continuous survey effort is significantly improved. The operation is more efficient; administrative staff members are more productive. Downtime is minimized, data quality is maximized. And ultimately, a smoothly running production system is well-suited for expansion.