Census Setup with Blaise 5 Manipula

Michael K. Mangiapane and Roberto Picha, U.S. Census Bureau

1. Abstract

In the process of CAPI and CATI data collection at the U.S. Census Bureau, Blaise is one of a number of parts that work together to ensure we successfully collect data. Before a survey is fully deployed for data collection using Blaise, it needs the appropriate data from our control and case management systems. One of the first steps in this process is to run a Manipula "setup" script that reads a data file and uses the file's contents to create individual Blaise databases. These Blaise databases are later loaded on an interviewer's computer to conduct a survey.

When Manipula became available in Blaise 5, we researched how we might use it to implement scripts that have the same functionality as our Blaise 4 scripts. We modified our setup script to run in Blaise 5 and tested its functionality versus the original Blaise 4 script. This presented some challenges since Blaise 5 creates and references databases differently than it did in Blaise 4. This paper will discuss how we overcame this challenge and others so that individual Blaise databases were produced by the setup script. We will also discuss other challenges in using Blaise 5 with the setup program and our progress in converting other Manipula scripts from Blaise 4 to Blaise 5.

2. Introduction

Blaise is one of a number of software programs used by the U.S. Census Bureau in the process of conducting surveys and collecting data. The software applications may be in different languages and exist on different platforms, but they must be able to talk to one another to pass the correct data needed to accomplish their functions. Blaise needs to be able to talk with our Case Management, WebCATI, and our Master Control systems, each one containing their own unique requirements. The way we accomplish this is by using Manipula to translate between Blaise and our other systems.

One of the early steps in our data collection lifecycle is to take a sample input file provided by our sponsors, known as a Sample Control Input File (SCIF), load the data and create individual Blaise databases. These databases are assigned and transmitted to laptops in the field or pushed down to desktop computers in our telephone centers. To accomplish this we create a "Setup" Manipula script which is a standard file created and packaged with all of our Blaise 4 surveys.

Now that Blaise 5 has added Manipula to their suite, moving a survey into Blaise 5 becomes easier for the Census Bureau. We have begun researching how our current Setup script and other Manipula scripts fit in with Blaise 5 and what changes are needed to conduct our surveys. This paper will talk about our current efforts to update the Setup script and other survey related scripts to make them Blaise 5 compatible and the updates required to the code. We will also look at a few challenges that were presented during this process, what we did to work around them, and future plans for our usage of Manipula. As of this paper, we are using Blaise 4.8.4.1861 for most of our current production surveys and Blaise 5.0.5.950 for our current research.

3. Current Blaise 4 Setup

In our current surveys environment, the survey sponsors provide a SCIF that is processed and used by the survey control systems and instrument. A typical SCIF is an ASCII text file that contains "record typed" data that is read in by the appropriate system. Each system (including the instrument) processes only the data that it requires. The control systems use record types that correspond to survey level settings, address information, and contact information. The instrument uses many of these record types along with survey (interview) specific information. Each line of the SCIF corresponds to a "record type" which is like a map that tells our Setup program where to read the data. Most of these record types are standard across our surveys and are identified by a four-digit number that indicates the record type and subtype that is being used for that survey. One exception to the

standardization is the 8500 series of record types. They are known as “dependent data” as they contain large amounts of survey-dependent data so these records are customized for each survey.

In our Blaise 4 Setup, we define each record type as a datamodel, entering the provided variable names and length of each variable so that they correspond to the position of the data for the record. Each datamodel is available as a part of the input file definition, with the output being a Blaise database:

Listing 1. Datamodel Source Code

```
INPUTFILE
    In1060  : InitRT1060   ('lqiinput.dat', ASCII)

ALTERNATIVE
    In2060      : InitRT2060
    In2561      : InitRT2561
...
```

Another output file produced by the Setup program is a list of Case IDs, which are unique identifiers that correspond to each case. These come from the SCIF and it serves as a record of which cases were created. We use this for verification that all of the expected cases were created for the interviewing period.

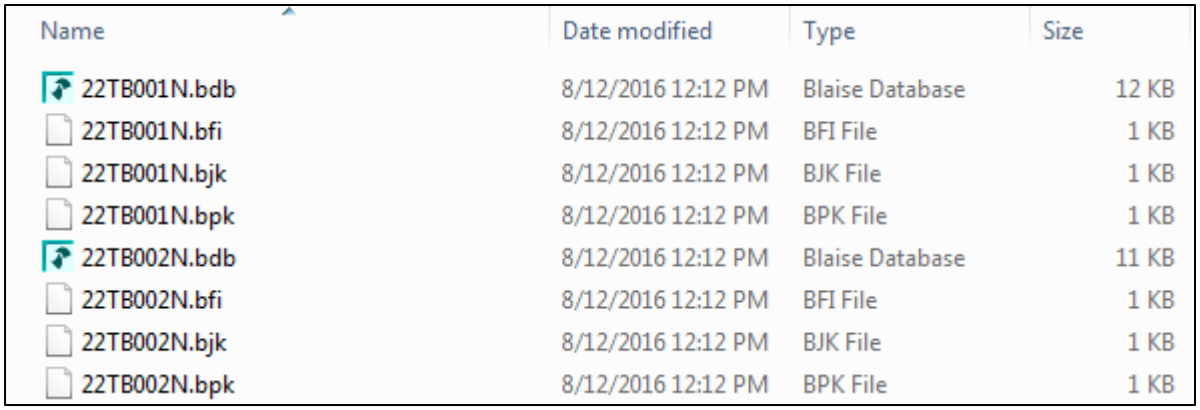
As the Setup script runs, it reads in each line of the SCIF. The first two characters correspond to the record type series (10, 20, 25, etc.) so the script matches which record has been read in and reads the appropriate datamodel to load the script. Each new case starts out with Record Type 10 so that is used as a signal to the script to create a new database to load data at the top of the script. Once that is done the script follows its standard procedure to load data into the case. It will perform a block to block copy of the data into a block at the datamodel level of the instrument (for backup purposes) and it will also load data into individual variables defined in the instrument.

Listing 2. Sample of Blaise 4 Manipula Script.

```
IF In1060.RecType = '10' THEN
    {If not the first record in the file }
    IF ID <> In1060.Rt1060in.CASEID  THEN
        OutFile.OPEN(In1060.Rt1060in.CASEID)
        Outfile.INITRECORD
        {Copy Record to Block}
        OutFile.Rt1060 := In1060.Rt1060in
        {Copy to datamodel level variables}
        Outfile.CASEID := In1060.Rt1060in.CASEID
        Outfile.CTRLNUM := In1060.Rt1060in.CTRLNUM
...
{Determine record type and load appropriate data}
ELSEIF In1060.RecType = '25' THEN
    {Copy Record to Block}
    OutFile.Rt2561 := In2561.Rt2561in
    {Copy to datamodel level variables}
    Outfile.PRICEEDIT := In2561.Rt2561in.PRICEEDIT
    Outfile.SALEPRICELO := In2561.Rt2561in.SALEPRICELO
ENDIF
In1060.READNEXT
...
```

The script continues in this manner until it reaches another Record Type 10 line in the SCIF. At that point the script writes to the Blaise database, closes it, and starts the process again until it reaches the end of the SCIF. The result is a set of Blaise databases that are ready to be packaged up and assigned to the appropriate computers to conduct the survey.

Figure 1. Blaise 4 databases created by current Setup script, Cases 22TB001N and 22TB002N.



Name	Date modified	Type	Size
22TB001N.bdb	8/12/2016 12:12 PM	Blaise Database	12 KB
22TB001N.bfi	8/12/2016 12:12 PM	BFI File	1 KB
22TB001N.bjk	8/12/2016 12:12 PM	BJK File	1 KB
22TB001N.bpk	8/12/2016 12:12 PM	BPK File	1 KB
22TB002N.bdb	8/12/2016 12:12 PM	Blaise Database	11 KB
22TB002N.bfi	8/12/2016 12:12 PM	BFI File	1 KB
22TB002N.bjk	8/12/2016 12:12 PM	BJK File	1 KB
22TB002N.bpk	8/12/2016 12:12 PM	BPK File	1 KB

4. Creating a Blaise 5 Setup

Since Blaise 5 Manipula contains significant changes in its language, our Blaise 4 scripts must be updated to make them compatible with Blaise 5. In the current Blaise 5 version of Manipula we do not have the ability to do a block to block copy, which was functionality we wanted in our Setup script along with copying data to our datamodel level variables. That was an immediate challenge for running our Blaise 4 scripts, but we were able to come up with an alternate way to load our data and keep that functionality.

One method that is available for both Blaise 4 and Blaise 5 Manipula is the PUTVALUE method. It allows us to feed in a fully qualified field name in the instrument and the value that should be placed in this field, including a Don't Know or Refusal. We decided as a part of the design process to use PUTVALUE to place our data because not only could we feed in the full path to the block where we keep the backup copy of the data, we could call it again and use a substring to copy the data to the datamodel-level variable. Since there is only one block name before the variable name, we can use the POSITION function to find the position of the period that separates the two, resulting in the variable name only. For example:

Listing 3. Position Function Source Code

```
iPos := position('.',iPath)
iLen := LEN(iPath)
//load variable at the datamodel level
Outfile.PUTVALUE(SUBSTRING(iPath,(iPos+1), (ilen-ipos)+1), Mydata)
//load variable at the RT block level
Outfile.PUTVALUE(iPath, Mydata)
```

In a similar fashion to our Blaise 4 Setup, we coded the script to read in each line of the SCIF until it reaches a Record Type 10 and then it writes out the data to the database. This became an opportunity to simplify coding the script and it makes it easier to do later maintenance if any record layouts are ever changed. Each input variable is written into the backup block and into the datamodel-level variable in the instrument so PUTVALUE must be called twice. We could just simply put a call to PUTVALUE for every variable in the instrument but that leaves us with a very long script. Many of our instruments are reading in over one-hundred twenty (120) variables from the standard record types and dependent data could more than double that number.

A best practice in coding is if there is something that is done repeatedly but not in a loop, then place it into a procedure. Rather than call PUTVALUE twice, we could just call the procedure once for each variable. If any records on the SCIF change, adding or removing a single procedure call rather than removing both instances of PUTVALUE is easier for maintenance, especially with larger surveys. It also means changing a single procedure if extra functionality needs to be added or removed instead of changing nearly every line of the script.

Listing 4. Procedure Source Code

```
prc_FormatData('r','rt1060.CASEID', Substring(RTLine[1],5,8))
prc_FormatData('r','rt1060.CTRLNUM', Substring(RTLine[1],13,24)
```

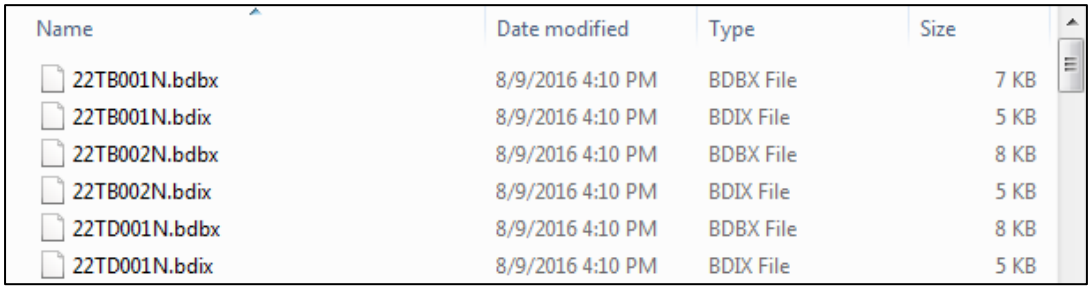
Another design consideration we added in to our script was to make an array of strings, with each element corresponding to a single record type on the SCIF. This allows all of the data for the SCIF to be loaded and we could examine individual lines in the script if we need to do any debugging. It also makes our script more readable since a programmer can easily spot the variable(s) they are looking for when reviewing and updating the code in the future.

Listing 5. Array of Strings Source Code

```
IF Substring(In1002.OneLineRT,1,4) = '1060' THEN
RTLine[1]  := In1002.OneLineRT
ELSEIF Substring(In1002.OneLineRT,1,4) = '2060' THEN
RTLine[2]  := In1002.OneLineRT
```

The resulting Blaise 5 Setup performs like the Blaise 4 Setup so it will be familiar to our programmers. In the MANIPULATE section the script reads in the individual lines of the SCIF and places each one in an array until it reaches the next record type 10. At that point it verifies that the Case ID on that record type 10 is a different case ID than what is already held. If it is different then the script calls a procedure to load the variables into the database. Inside the load procedure, we create an empty Blaise database and then call a procedure that takes each variable from the SCIF and uses PUTVALUE to place them in the appropriate instrument variables. Once the load procedure has gone through all of the variables, it empties out the array and starts reading the next set of records and loading them into the next database. This process is repeated until it reaches the end of the file. This results in a set of individual Blaise databases and data interface files

Figure 2. Blaise 5 Databases created by the new Setup script. Case ID's 22TB001N, 22TB002N, and 22TD001N.



Name	Date modified	Type	Size
22TB001N.bdbx	8/9/2016 4:10 PM	BDBX File	7 KB
22TB001N.bdix	8/9/2016 4:10 PM	BDIX File	5 KB
22TB002N.bdbx	8/9/2016 4:10 PM	BDBX File	8 KB
22TB002N.bdix	8/9/2016 4:10 PM	BDIX File	5 KB
22TD001N.bdbx	8/9/2016 4:10 PM	BDBX File	8 KB
22TD001N.bdix	8/9/2016 4:10 PM	BDIX File	5 KB

You can view a snippet of the Setup script code in Appendix A.

One major advantage to the Blaise 5 Setup is how fast it runs when compared to the Blaise 4 script. Using a test input file that creates 74 cases, the Blaise 4 Setup takes approximately 1 minute and 10 seconds before it finishes executing. The Blaise 5 Setup takes approximately 12 seconds. If we extrapolate this speed difference to a number of survey cases for a national survey, like 10,000, it will take approximately 2 hours and 15 minutes for the Blaise 4 Setup to process those cases. The Blaise 5 Setup should take approximately 27 minutes to process the same number of cases. This very large difference will have an impact on the speed of our processing systems. As we move further with testing Blaise 5 instruments we will know more about how much of an impact it will have.

4.1 The Blaise 4 to Blaise 5 Setup Conversion Script

We have the tools available in Blaise 5 to convert our Blaise 4 datamodels, data files, and even Manipula scripts, which saves a lot of time in converting instruments to Blaise 5. However, since we were changing a large portion of the code used in the Setup script, we built our own Blaise 4 to Blaise 5 Manipula conversion script written in Blaise 4 Manipula. This script reads in a Blaise 4 Setup script

and turns it into a Blaise 5 compatible script. With each record type defined as their own block, we used the GETFIELDINFO to find the name of each block, the name of each variable in the block, and the size of each variable as they are needed for the Blaise 5 Setup.

The conversion script starts by creating a setup.manx file for output. It prompts the user for the number of dependent data records that are needed for their script since these vary by survey.

Figure 3. User prompt for the number of 8500 records in the current SCIF.

Next, a "header" that contains the first lines of the Setup script is added, including defining an INPUTFILE and OUTPUTFILE. This continues until it reaches what would be the loading procedure for the Setup script. After that our conversion script searches the Blaise 4 database for any blocks that start with "BRT" or "BRT85" as those are standard names for our record type blocks in our instruments. When it finds the block, it stores the block name into an array of names. After that search is done the script calls a procedure to pull the name of the variables and characteristics of each variable for each block that was stored in the array. As it pulls each name, the procedure writes out the characteristics with other string constants to produce the lines that call the procedure to copy data into the Blaise 5 database. Once all of the fields have been written to the new script, the "footer" is written. The "footer" contains the rest of the Setup script, including the MANIPULATE section.

Blaise 4:

Listing 4. Blaise 4 Source Code

```
Outfile.COLLECT_START      := In2561.Rt2561in.COLLECT_START
Outfile.COLLECT_SALES      := In2561.Rt2561in.COLLECT_SALES
Outfile.COLLECT_COMP       := In2561.Rt2561in.COLLECT_COMP
Outfile.COLLECT_MCDNOTES   := In2561.Rt2561in.COLLECT_MCDNOTES
Outfile.COLLECT_FRREQUEST := In2561.Rt2561in.COLLECT_FRREQUEST
```

Blaise 5:

Listing 5. Blaise 5 Source Code

```
prc_FormatData('r','rt2561.COLLECT_START',
Substring(RTLine[3],1361,1))
prc_FormatData('r','rt2561.COLLECT_SALES',
Substring(RTLine[3],1362,1))
prc_FormatData('r','rt2561.COLLECT_COMP',
Substring(RTLine[3],1363,1))
prc_FormatData('r','rt2561.COLLECT_MCDNOTES',
Substring(RTLine[3],1364,1))
prc_FormatData('r','rt2561.COLLECT_FRREQUEST',
Substring(RTLine[3],1365,1))
```

The conversion script removes a lot of the time required to convert our Setup script from Blaise 4 to Blaise 5, though the user does still have to enter some variables manually. We left off pulling the dependent data variables and placing them into the Setup script at the time of the conversion because

not all of our surveys copy the dependent data into their own blocks at the datamodel level of the instrument. Most of our surveys directly copy dependent data into their matching variable in the instrument so it makes more sense to have the user supply the fully qualified variable names as they can pull those from the Blaise 4 Setup script.

4.2 Setup Script Challenges

One of the immediate challenges that was noted earlier in this section was being able to replicate some of our Blaise 4 script functionality in Blaise 5. Manipula is a fairly recent addition to the Blaise 5 and much like Blaise 5, it is being completely overhauled to take advantage of new technologies and additional capabilities that are now available in Blaise 5. Since it is a work in progress, not everything that we had in Blaise 4 Manipula is available.

4.2.1 Maniplus Not Available

One significant feature not available yet in Manipula is Maniplus. This means that commands such as RUN and FILEEXISTS are not available for us to call out to processes that rename files. Besides these commands, many of the Manipula scripts that we use in our survey instruments to perform functions like calling the instrument, sampling, unduplication of data, and some navigation within the instruments are not available for use in Blaise 5 yet. This affects a number of our surveys that use these Maniplus scripts.

4.2.2 Block to Block Copy

As mentioned earlier in "Creating a Blaise 5 Setup," the current version of Manipula does not allow for a block to block copy. We were able to find a workaround by using procedures to tell the Setup script to directly place each variable value into the record type blocks in our instrument. We could simplify our Setup script if we had the ability to use a block to block copy since only one line of code would be needed to copy all of the values into the block versus a line of code for each variable. It would shorten our script as well, making it easier to maintain.

4.2.3 Calling Cases Individually

A challenge that is currently under research is the ability to call a specific case (individual BDB) to run and store data. Blaise 5 surveys expect to connect to a central database to retrieve and store their data, with each case being a record in this central database. We have noticed that when we call the DEP, it expects to find one database to connect and retrieve its record, and we cannot specify which database file that Blaise 5 should connect to in order to retrieve the form and run the case. There are a few potential options that we could consider to work around this, including creating a single Blaise database on each laptop as a placeholder where each case could be placed. That option may or may not be available since it creates the potential requirement that all of our control systems and our case management software would need to be updated to handle setting up one Blaise database. Another option would be to have a central database on a server where laptops could connect and retrieve individual cases. However, that is not entirely feasible as we do need our surveys to run in offline (disconnected) mode due to the potential for interviews occurring in areas that have no steady connection to the Internet. We will continue to investigate options for integrating Blaise 5 into our existing control systems.

4.2.4 Other Commands Not Available

At the time of originally coding the new Setup script (5.0.4.875) items like the ALTERNATIVE section and AUTOREAD were not available. Some of these missing features have appeared in Blaise 5.0.5.950 and more will be added. However, our finding of alternatives such as reading each line of the SCIF into an array and parsing the array to load data has resulted in a leaner script that is more efficient. As Manipula is enhanced for Blaise 5, we may find even better ways to run our Setup than what we have now.

5. Converting Other Manipula Scripts

There are three other standard Manipula scripts used in our CAPI interviewing process. These are known as our Case Management In, Case Management Out, and CAPI_trans scripts. The Case Management In script takes information passed from our Case Management software such as the respondent's contact information and updates the appropriate variables in the instrument. The Case Management Out script does the opposite in that it passes information such as updated respondent contact information and outcome codes from the instrument back to our Case Management software. The CAPI_trans script is what calls the Case Management In and Out scripts along with calling the instrument or updating instrument data based on the transaction code that is passed by our Case Management software.

With the success in converting the Setup script, we decided to convert the Case Management In and Case Management Out scripts to Blaise 5 so that we could test their functionality. The Case Management In script is like a simpler version of the Setup script. It receives an ASCII file that is much like a SCIF except all data is on a single line and there is no record type needed. It also opens an existing Blaise database so a new database does not need to be created on the fly; it just has to update the appropriate variables in the instrument. We were able to take the same approach used in our Setup script in that we read in our input file, parse out the data in the appropriate position, and update the instrument variables. In the future, we plan to use an XML file instead of an ASCII file for this process.

The Case Management Out script is still a work in progress. We have been able to create a script that compiles and runs with our Blaise 5 instruments, but it only outputs the XML tags that Case Management uses; it does not output any of the instrument data. Only blank spaces are output where the data would go. We are researching to determine if this is due to the script not actually pulling the data from the case or if it is not writing it to the output file. We are confident we will figure out how to retrieve the data from a Blaise 5 database and put it into a custom output file.

The CAPI_trans script currently needs some of the functionality that is only available in the Blaise 4 Maniplus, such as the CALL command to call other Manipula scripts like the Case Management In and Out scripts and the EDIT command to run the DEP. We are researching how to work around those limitations.

6. Future Plans

We have had some great success so far with our research into Blaise 5 Manipula and converting our scripts, but we are just getting started. We have proven so far that we can make our scripts work and potentially work better, but we always like to push it further to get the most out of Blaise and Manipula.

In regards to the Setup script, there are a few surveys that do things a little differently that will need to be updated if we are going to be able to move them into Blaise 5. For example, the SCIF for the Consumer Expenditures Quarterly (CEQ) survey has cases that have multiples of the same dependent data records, such as for families with multiple vehicles or who pay multiple utilities. Per the current design of the Blaise 5 Setup script, if we had a household with two vehicles it would write the first vehicle into the array and then we would overwrite that array element with the second vehicle. So for CEQ's Setup script, we may modify it to capture the full record type for each used element in the dependent data array and use that to call a procedure to pick the set of instrument variables to load the data. Our Blaise 4 to Blaise 5 Setup conversion script will also be modified to collect some additional information since it may be possible to automate some of the dependent data in the Setup.

A number of our surveys have Manipula scripts that are called during a DEP session. These include scripts that perform sampling functions or that do data quality checking such as finding duplicate data. Those scripts will need to be converted and tested to make sure they retain the same functionality that

they had in Blaise 4. It is expected that we will be able to add new functionality and our scripts will continue to evolve as Manipula evolves.

7. Conclusion

Manipula is a valuable tool in Blaise as it allows us to extend our capabilities with our survey instruments. The Census Bureau has been able to take advantage of its features for years and we continue to discover new ways to improve our instruments using Manipula. As new features are added to the Blaise 5 Manipula there will be new possibilities to try something different and learn more about what our survey instruments can do. The future of our instruments and using Manipula reflect the hard work that has already gone in to creating and researching what Manipula can do now. We have had success so far in our Blaise 5 Manipula research and there is still work to be done. It makes for interesting times as survey instruments and Blaise evolve.

8. Acknowledgements

The author would like to acknowledge the work and knowledge of Mecene Desormice as his assistance in converting Setup scripts from Blaise 4 to Blaise 5 along with converting other Manipula scripts was invaluable in our research into using Blaise 5 Manipula.

The views expressed in this paper are those of the authors and not necessarily those of the U.S. Census Bureau.

9. Appendix A - Blaise 5 Setup Script Code Snippet

```
//Format and load the data into the instrument
PROCEDURE prc_FormatData
   PARAMETERS IMPORT  iFlg,iPath,iVal: STRING
   AUXFIELDS iPos, iLen : INTEGER
   Mydata : STRING

   INSTRUCTIONS
      Mydata:= Empty
      Mydata := Trim(iVal) //Trim all trailing spaces
      IF iFlg = 'r'   THEN
         iPos := position('.',iPath) //Find the position of the
variable name
         iLen := LEN(iPath)
         //load variable at the datamodel level
         Outfile.PUTVALUE(SUBSTRING(iPath,(iPos+1), (ilen-ipos)+1),
Mydata)
         //load variable at the RT block level
         Outfile.PUTVALUE(iPath, Mydata)
      ELSE
         Outfile.PUTVALUE(iPath, Mydata)
      ENDIF
ENDPROCEDURE

PROCEDURE prc_Load
   INSTRUCTIONS
//Open a bdbx or create it if it doesn't already exist
      Outfile.OPEN(Substring(RTLine[1],5,8) + '.bdbx')
      Outfile.INITRECORD
//Call the procedure to load all of the data
      prc_FormatData('r','rt1060.CASEID', Substring(RTLine[1],5,8))
      prc_FormatData('r','rt1060.CTRLNUM', Substring(RTLine[1],13,24)
      ...
      prc_FormatData('r','rt2060.ADDR1', Substring(RTLine[2],5,54))
      prc_FormatData('r','rt2060.ADDR2', Substring(RTLine[2],59,54))
```

```
      ...
      //Write the data to the instrument and close the database
      Outfile.WRITE
      Outfile.CLOSE
ENDPROCEDURE

MANIPULATE
      //Read the next line of the input file
   In1002.READNEXT
      Verify_ID :=  Substring(In1002.OneLineRT,5,8)
      //Read each line of the file until you hit the end of the file
      WHILE NOT (In1002.EOF) DO
      //Match to the record type and fill the appropriate array
element
      IF Substring(In1002.OneLineRT,1,4) = '1060' THEN
          RTLine[1]  := In1002.OneLineRT
      ELSEIF Substring(In1002.OneLineRT,1,4) = '2060' THEN
          RTLine[2]  := In1002.OneLineRT
      ...
      ENDIF
      //After reading into array, read the next line
      In1002.READNEXT
      //If the next line starts with 10, it's a new case
      IF Substring(In1002.OneLineRT,1,2) = '10'  THEN
         IF Substring(In1002.OneLineRT,5,8) <> Verify_ID THEN
            //Load the data stored in the array into the instrument
prc_Load
//Update the verify ID so it only reads in the current case
            Verify_ID := Substring(In1002.OneLineRT,5,8)
         ENDIF
      ENDIF
   ENDWHILE // NOT (In1002.EOF)
```