Listing Part 2 - Using Blaise 4.7 for Coding Listing Instruments[*]

Malcolm Robert Wallace, Roberto V. Picha & Michael K. Mangiapane (U.S. Census Bureau)

1. Overview

The U.S. Census Bureau conducts many listing operations using computer assisted interviewing (CAI) instruments (referred to as listing instruments), as a means of obtaining sample for many of the surveys that it conducts. At the 2004 IBUC, the Census Bureau presented a listing paper that discussed different listing operations conducted by the Census Bureau, the unique challenges and functionality required by listing instruments, and how some of these requirements were addressed with Blaise 4.5. Since the 2004 IBUC, the Census Bureau has begun work on converting our two oldest and most challenging listing instruments – Permit Address Listing (PAL) and the Survey of Construction (SOC). This paper will discuss how the Census Bureau plans to use Blaise 4.7 to address some of the more challenging functionality required by these two surveys.

1.1. Paper Objectives

This paper will provide some background on the PAL and SOC listing operations, review some of the listing requirements requested, and discuss how we have addressed (or plan to address) these requirements using Blaise 4.7. This paper also includes some code examples.

1.2. What is a Listing Instrument?

A listing instrument is a CAI instrument that is used as a tool by the Field Representatives (FRs) to collect, and sometimes sample, lists of data in order to generate sample cases for interviewing. These lists of data can be various things, such as lists of permits, addresses, or names. Once the listing is complete, sample is either generated by the listing instrument, or back at Census Headquarters (HQ) after the listing data transmitted in.

1.3. Permit Address Listing (PAL)

The PAL survey is a monthly listing survey that has been in production in CAPI since 1996. The PAL instrument collects building permit information for new residential construction. The FRs visit permit offices each month and key in the permit numbers, address information, and geocode information for every residential permit issued that month. This is a "true" listing instrument in that there is very little "interviewing" done with this instrument. The FR simply enters the listing instrument and starts keying data.

The data collected by the instrument is used to update the new construction sampling frame for the Census Bureau's current surveys.

Although it sounds simple, this is a fairly complex instrument behind the scenes. There are many special situations that the instrument must account for. There is also a lot of added functionality to aid the FRs in their listing.

[*] ***Disclaimer:*** *The views expressed in this paper are those of the authors and not necessarily those of the Census Bureau.*

1.4. Survey of Construction (SOC)

The Survey of Construction is a monthly listing and interviewing survey that has been in production in CAPI since 1995. The data from SOC is used to produce the monthly housing starts statistics. The SOC listing instrument also collects permit information from building permit offices. It gathers the permit number, address information, and builder information. In addition to doing a complete listing of the permits for the month, this instrument also conducts complex sampling of these permits and generates input files that are used by another CAPI instrument. Once the listing is complete, the FRs conduct interviewing using this other interviewing instrument.

1.5. PAL and SOC Redesign Efforts

Both the PAL and SOC listing instruments are currently written in Clipper and must be converted to Blaise. Since these two operations are similar, in that they both collect lists of data from building permit offices, we decided to combine their conversion efforts together and discuss requirements for both at the same time. Although both of these listing applications involve the listing of permit information, they also contain very unique and survey-specific functionality that did not allow us to combine them into one instrument. For example, the SOC listing instrument (LI) requires a complex sampling algorithm that is not needed by PAL. It also lists a lot of information that is not needed by PAL – such as builder contact information and specific information from the permit. PAL on the other hand requires two types of listing tables – one that lists all of the permit information and one that has some information already pre-filled which then requires special editing and handling.

Sponsors of both surveys were brought into the same workshop so common requirements could be addressed together.

1.6. Requirements Workshop

We held a week long requirements gathering workshop. The workshop participants included people from the Technologies Management Office (TMO) authoring and case management systems staffs, the Field staff (SOC/PAL supervisors, a senior Field Representative (SFR), and HQ staff), and the sponsoring divisions. During the workshop we reviewed the current system and defined the requirements that would be needed for the new, redesigned system. This included adding new functionality to the system that the sponsors/Field wanted and removing functionality that was no longer needed. It also included modifying existing functionality in order for it to work with Blaise. We also made an attempt to standardize similar requirements for both instruments. For example, we agreed upon standard field lengths for common fields and the address format for the permits and building permit offices (BPOs). Requirements and outstanding issues were documented and distributed to the design team. These requirements were also used as input into the case management systems requirements workshop that was held at a later date.

2. Functionality Requested for the SOC and PAL Listing Instruments

The concept of a listing seems quite simple. One would think that you could just create a table and key in some information. Of course, nothing is ever as easy as it sounds (or as it should be). This section documents some of the different requirements requested by our sponsors and Field staff.

2.1. Combine Questionnaire into Listing Instrument.

The current SOC listing operation uses 4 different instruments:

1. Listing Instrument (LI) – the Clipper program that collects the permit information
2. Questionnaire for Building Permit Officials (QBPO) – CASES instrument that collects information about the permit office. For example, it collects the location of permits, permit filing information, and information on the BPO boundary.
3. Little Questionnaire Instrument (LQI) – CASES instrument that collects more detailed information from the permit. For example, the lot size, sale price, number of rooms, etc.
4. Builder Table (BT) program – Clipper program that helps the FR manage their builder information

For the SOC redesign we are able to combine three of these programs into a single Blaise listing instrument – the LI, QBPO, and LQI. Since all three instruments are conducted at the permit office it makes sense to combine them into one program. This is one of the advantages of using Blaise 4.7 for the SOC LI – we can include a questionnaire within the LI.

2.2. Call an External Program and Insert Data Into Listing

During the SOC listing process, the FRs must collect the builder name, address, and contact information for each listed line (building) that falls into sample. To help the FRs keep track of the builders in their assignment area, they store the name and contact information in a "Builder Table" which acts like an address book. It is basically a database with the contact information the FRs need to complete the interviewing portion of their work. Instead of having to key in the builder name, address, and contact information for every sample line, the FRs only need to key this information once (into their Builder Table) and then select a builder contact from the BT and have the instrument automatically fill the data for these fields. This is functionality that they currently enjoy and must continue to have available when we convert to Blaise. The other requirement with this application is that the case management system also needs access to the BT database so that program can get the most up-to-date information as well.

This paper will discuss how this functionality was addressed with Blaise 4.7.

2.3. Creating Look-up Tables on the Fly

Since PAL and SOC both involve collecting addresses for new construction, many of the street names keyed do not already exist in some type of street name file for the BPO. However, the sponsors would still like to have a street name pick-list for the FRs to use during the listing. They want this street name pick-list to contain all of the street names listed for the month. Since builders often take permits out in groups, it is likely that there will be multiple permits issued for the same street name. The sponsors would like the FRs to use this street name pick list so that street names are listed consistently throughout the listing.

Similarly, the sponsors have requested a ZIP Code/Locality pick-list also be created "on the fly" so that ZIP codes and localities are listed consistently.

This paper will discuss how this functionality was addressed with Blaise 4.7.

2.4. Toggle between “Line View” and “Full View”

When thinking of a listing table, one thinks of the “Line View” display. This display simply collects data one row after the other (with many rows displayed on the screen at the same time) with columns of the row scrolling off to the right of the screen. Since the PAL LI collects a large amount of data, the sponsor has requested that we allow the FRs the option to enter their data all on one row (“Line View” with scrolling and many rows on the screen at one time) or to enter their data all in one screen, a “Full View” that displays all of the data for one permit with no scrolling.

This paper will discuss how this functionality was addressed with Blaise 4.7.

2.5. Using Different Listing Tables Based on FR’s Preference

For the SOC LI, the FRs must list “additional information” for listed lines that fall into sample. This information includes the address of the permit, the builder / contract information, and some details on the structure (from the permit). Since permit offices collect and store their permit information in different ways, the sponsor requested that we give the FRs two options for collecting the additional information:

1. Collect additional information while listing. This means that all of the fields come on route if the permit falls into initial sample. The FRs would choose this option if it would be hard for them to go back and find the permit information after final listing is complete and sampling is run.

2. Collect additional information after listing is complete and final sampling is run. This means these fields do not come on route during initial listing. The FRs would pick this option if it would be fairly simple for them to go back and gather the additional information needed for sample permits. This option would require less keying by the FRs since Final sampling will screen out some of the initially sampled permits.

This paper will discuss how this functionality was addressed with Blaise 4.7.

2.6. Free Form Navigation

Blaise 4.7 Free Form Navigation (FFN) has been very useful in our design of the PAL listing instrument. It has helped with many of our navigation requirements. Listed below are some of the navigation requirements we have for our listing table along with how we are planning to handle these with Blaise 4.7.

- **“Force” some Fields to be “Must Enter” in the FFN Table.** Ideally, we would like to be able to control whether the fields “must enter” requirement is enforced while in the FFN table. For example, every listing line in the table must have a permit number or there would be no reason to have that record. This currently can’t be done with the Blaise FFN (and may not make sense to do), but we have a work around for this requirement. To address this particular requirement for PAL, we do not bring the rest of the listing line on route until a permit number is keyed in.

- **Run Field Edits when data is entered into a Field**. If data is entered into a field, then the edits on this field should be run at that time. So, if the FR enters in an invalid date, the date range edit should pop immediately. (Blaise 4.7 now does this.)

- **Run “Error Involving” Edits when data is entered into a Field**. As with regular edits, edits on multiple fields specified in an “error involving” edit should be run when the data for all fields involved is entered. For example, if the FR enters a month = June and day = 31, the edit check should run and pop an error as soon as "31" is entered in the "day" field. (Blaise 4.7 now does this.)

- **Allow for "must enter" fields to remain empty when exiting the FFN table**. One of the functional requirements for the PAL LI is to allow the FRs to key in partial permit data and exit the listing table (without popping an error). For example, it would be acceptable for the PAL FR to list the permit information, but not complete the ZIP code/locality information until a later time (maybe at home) since the permits may not contain this information. This means the FR would have to exit the case, leaving the data unfilled and the case as a partial, and then re-enter the case and complete the listing.

  Blaise 4.7 allows us to map a function key (F10 in our case) that takes the instrument to a parallel block (the back of the instrument) and will not enforce the "must enter" requirement for fields when doing this. This is all right since we would code this case as a partial when the FR exits in this manner. In order to complete the listing, we plan to have the FR exit through a "listing complete" screen which will then verify "must enter" fields are completed by using Maniplus to run edits on them.

Blaise 4.7 FFN will automatically enforce the "no empty" check if the user exits the table by completing the last possible line in the table or when moving to a new page. In this case, we would like Blaise to allow us to have some control over the error messages that are displayed to the FR in the Dialog box.

2.7. Exiting the Listing Table Instrument

This seems like a simple item, but of course there is a catch. Our current standard for exiting a table is to have the FR enter "999" in the first column of the last row. This doesn't work for listing since "999" could potentially be a valid entry for the permit number field. Since Blaise 4.7 allows us to create button bars, we went ahead and created a "Done" button for the FRs to use when they are finished with the listing. This button is associated with an action that will start a parallel block in the back and take the FRs to an "Is your listing complete" question. If they answer "yes", we call a Maniplus script that will run edits on the listed data to verify all "must enter" fields are complete and to verify the information listed passes various form-end edits. If they answer "no", we return them to the listing table.

3. Calling an External Program and Inserting Data into Listing

Blaise lookup table capabilities provide an excellent way to call external files and insert data into the instrument. It is fast, reliable and easy to use. However, in order to meet our sponsor's requirements for their Builder Table (BT) application, we need to be able to interactively add, edit, and select information from this external file. We must do all these operations while inside the DEP. Since this Blaise external file does not provide the desired flexibility, we had to figure out another way to accomplish our sponsor's requirements.

Blaise 4.7 is a more robust version than its predecessor. Data operations generally done outside the DEP are now possible while in it. Blaise 4.7 now provides us a way to achieve the sponsor's requirements by allowing us to use a combination of Manipula and/or Maniplus that will enhance Blaise capabilities for data collection.

The biggest challenge we have with the SOC listing operation is the interaction between the listing instrument and the builder table application. The BT application is a separate program that can be accessed from multiple instruments and from the laptop case management system. It would typically be called multiple times from within the LI while the FRs are collecting data. The BT data is sent down to the laptop as a separate input file and will be stored in an Oracle database.

3.1. Possible Approaches

We discussed 3 possible approaches for addressing this requirement:

1. **Writing the BT Application in Blaise and Using ASCII Files -** This process would involve retrieving/updating data from the Oracle database into an external Blaise database and vice versa, by using ASCII files. Although this process could have been done in our current environment, we were concerned about the additional time needed for launching the instrument.

2. **Writing the BT Application in Blaise and Using BOI files** - This approach would be to use BOI to directly connect to the Oracle database, extract table information, and load it into a temporary Blaise database. Updates to Oracle database from the listing instrument can be done via BOI.

   Research concluded that this was possible; however, our prototype indicated that there was a significant delay connecting to the database (we used Access to mimic an Oracle database for our testing). This was a delay from within the DEP at the listing table and would not be acceptable for the FRs.

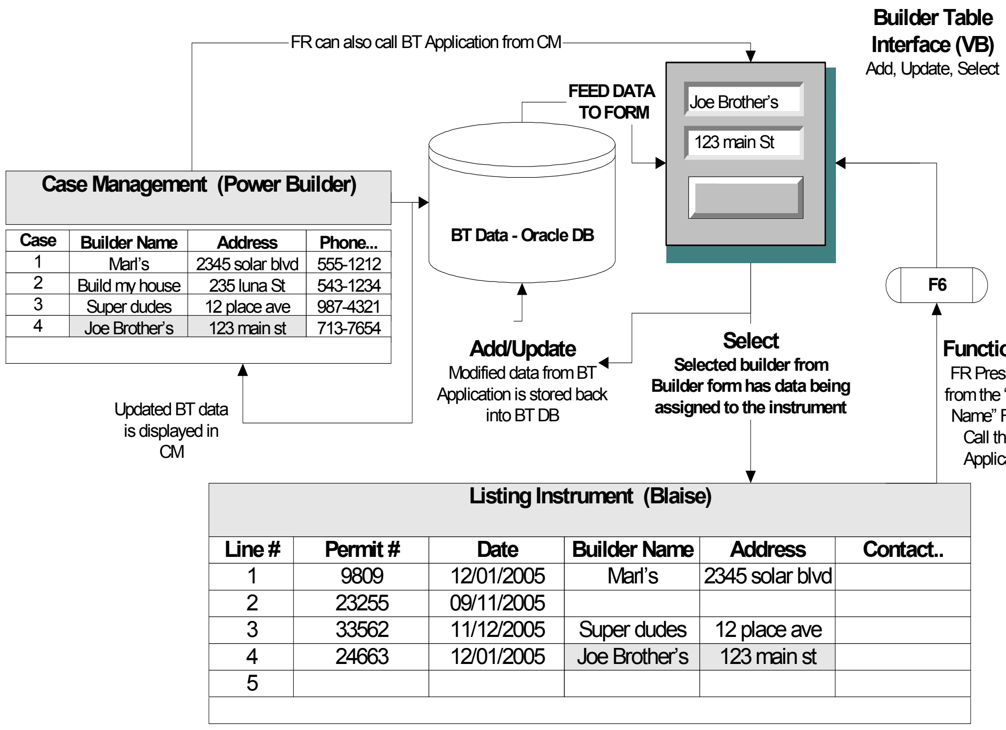
3. **Writing the BT Application in VB and Using the BCP -** For this approach, we plan to build a VB application that will connect to the Oracle database and allow user interface. This application will be called our Builder Table program and will allow users to add, edit, and select (transfer the data for the selected record down to the DEP – into the listing table). The BCP will be our mechanism to allow the DEP and VB program to interact.

   By using this approach, we accomplish two objectives: 1) allowing our case management system to interface with the BT data and 2) allowing two different instruments to interface with the BT data. Our initial testing indicates a fast response time.

   For us, this appears to be a feasible solution. For our case management system, this application can directly connect to the Oracle database on the laptop. For our instruments, the application can be "wrapped" via the BCP so the instrument can actually interact dynamically by transposing data to the DEP. This application would be converted into a wrapper DLL and installed on the laptops.

   The Builder Table interface, whether called from the case management system or the instrument, will retrieve information from the Oracle database and display it to the user based on parameters passed into the application. The updated information is then stored back to the Oracle DB so that the next time the application is called, the latest information is available for both case management and the instruments.

3.2. Data Flow of the Builder Table Application


FR can also call BT Application from CM
Builder Table Interface (VB)
Add, Update, Select
FEED DATA TO FORM
Joe Brother's
123 main St
BT Data - Oracle DB
F6
Add/Update
Modified data from BT Application is stored back into BT DB
Select
Selected builder from Builder form has data being assigned to the instrument
Function key
FR Presses F6 from the "Builder Name" Field to Call the BT Application
Updated BT data is displayed in CM

Case Management (Power Builder)

Case	Builder Name	Address	Phone...
1	Marl's	2345 solar blvd	555-1212
2	Build my house	235 luna St	543-1234
3	Super dudes	12 place ave	987-4321
4	Joe Brother's	123 main st	713-7654

Listing Instrument (Blaise)

Line #	Permit #	Date	Builder Name	Address	Contact..
1	9809	12/01/2005	Marl's	2345 solar blvd	
2	23255	09/11/2005			
3	33562	11/12/2005	Super dudes	12 place ave	
4	24663	12/01/2005	Joe Brother's	123 main st	
5					

4. Creating Lookup Tables on the Fly

When listing information for SOC and PAL, some information such as the city and ZIP code or full street name of an address may be the same for multiple lines of the listing. It would be time consuming to the FR if they have to re-type this information over and over again, especially if the city or street name is long. This could also lead to keying errors and inconsistencies with the data. For those reasons it would be useful if the user could use an external lookup table to enter the information into the listing.

4.1. Previous Lookup Tables

Previous versions of Blaise have allowed for a static external lookup tables to be defined and used in the instrument. Data could be passed from the lookup to the instrument, but data couldn't be passed from the instrument back into the table. We could potentially send out a lookup table with the instrument to the FRs, but there would be major problems with taking this type of approach. We would have to either send out an extremely large table with the instrument that could require a lot of time to search through to find the correct data, or we would have to tailor a lookup table for each FR that conducts these surveys. With thousands of CAPI assignments sent out across the country each month, tailoring a lookup table for each assignment is not feasible. The other problem is that this table would most likely need updating by the FRs since they are listing newly created streets. If the FR needs to add data to the table or update data that is already in the table, they could not do it themselves. Instead, some sort of update process would have to be created by HQ. If the FRs could not update the table, then it would not be as useful during their listing.

4.2. Using Blaise 4.7 for Dynamic Lookup

An idea that was proposed was to use a dynamic lookup table that could be changed by the FR. With the release of Blaise 4.7, a way to have such a table was created. By calling a Maniplus script that uses the INTERCHANGE setting for the DEP, it allows data to be passed from a currently running DEP session to another Blaise table. Since we can call a Maniplus script as a menu item from the DEP menu, a button in the DEP button bar, or as an action button in the field, it is now possible to have the dynamic lookup table we desire. It also reduces the amount of data we send out to the field because we can simply send out an empty table and let the FR fill it at their own discretion for the survey. Since we don't have to get the table back from the FR, it lessens the amount of data that comes back to headquarters that would require some sort of processing.

4.3. Creating the Table

In order to accomplish this task, a table needs to be created that will hold the data for the lookup. It is a simple process of creating a datamodel that is separate from the listing instrument with the fields you want to hold data inside the table. Make sure the field type inside the table exactly matches the field type in the instrument to prevent any errors in the data. This is especially important for string fields. After you compile and run the datamodel to create the Blaise database, the **minimum** files you need are the BDB, BDM, BFI, and BJK files to have a working lookup table.

See **Example 1** in the Appendix for a datamodel example.

4.4. Creating the Maniplus Script

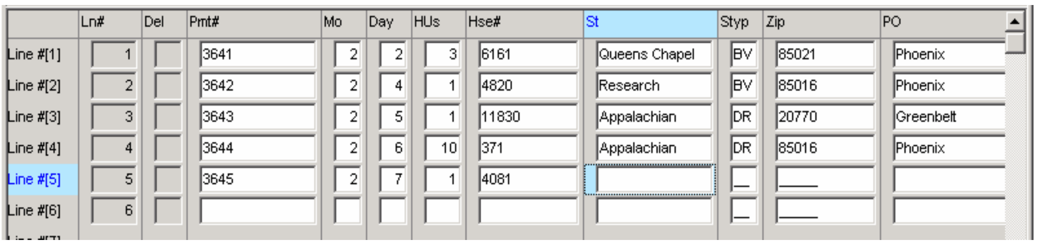
A Maniplus script is needed to tie the lookup table and instrument together. The nice thing about the INTERCHANGE setting is that it can be used for a TEMPORARYFILE, which is required to allow a DEP session to exchange data with the lookup table. Since we are using the current instrument DEP session, fully qualified field names from the DEP are allowed inside the script. You may share data directly between the DEP and the lookup table, or you may create auxfields to hold the data from the DEP before passing it on to the table. Choosing to use auxfields to hold the data is a good idea for debugging when working with Maniplus. It is recommended that the lookup table be set as an UPDATEFILE with the settings OPEN = YES, CONNECT = NO, and MAKENEWFILE = NO. This allows the table to be created and updated, and it will not erase all of the data when changes are made.

By using the ACTIVEFIELD, SUBSTRING, and POSITION functions in Maniplus we can pinpoint where (which field) in the listing the FR is at so that the script exchanges the correct data. Since the listing table in SOC is set up as an array of blocks, the field looks like this:

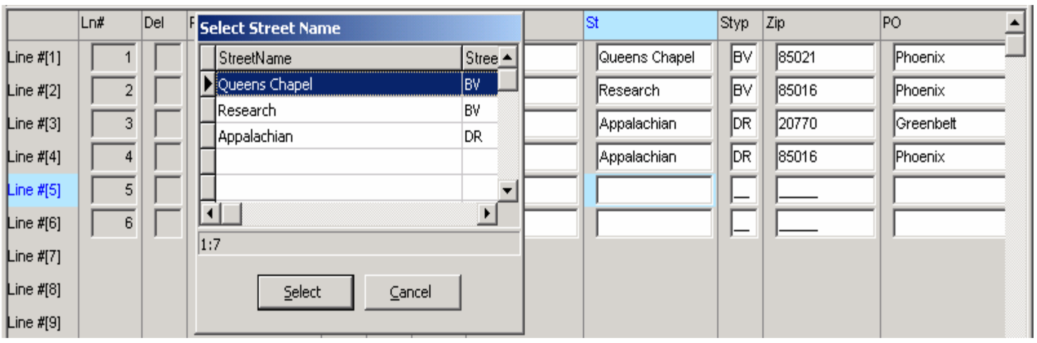
"TableName.BlockName[position].FieldName." We can use a substring that starts from the position + 2 of the right bracket to get the field name, and we can create another substring that finds the positions of the array position brackets and reads the text in between them to get the listing line the FR is on.

4.5. Retrieving Data from the Lookup

The FR will use a function key combination (Ctrl+L, for "list", in our case) to call the lookup table. A DIALOGBOX is used to hold the lookup table and it has two buttons, one to select the entry the FR decides to use and the other to cancel the lookup if they decide not to use any of the table data. When the user selects a line in the lookup table and presses select, the data in the table is read and passed to the fields specified in the script to the DEP.



	Ln#	Del	Pmt#	Mo	Day	HUs	Hse#	St	Styp	Zip	PO
Line #[1]	1		3641	2	2	3	6161	Queens Chapel	BV	85021	Phoenix
Line #[2]	2		3642	2	4	1	4820	Research	BV	85016	Phoenix
Line #[3]	3		3643	2	5	1	11830	Appalachian	DR	20770	Greenbelt
Line #[4]	4		3644	2	6	10	371	Appalachian	DR	85016	Phoenix
Line #[5]	5		3645	2	7	1	4081				
Line #[6]	6										

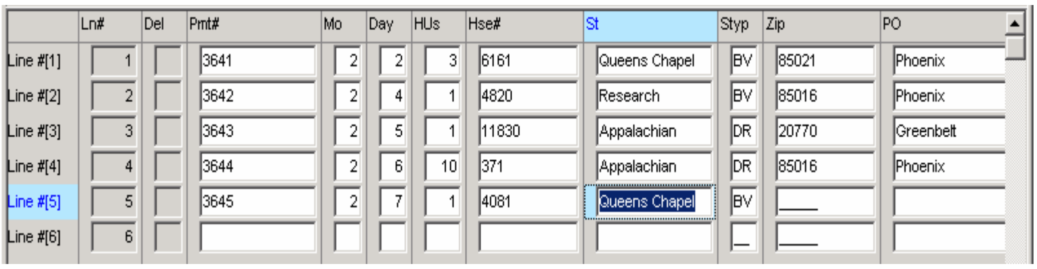


Select Street Name

StreetName	Stree
Queens Chapel	BV
Research	BV
Appalachian	DR

1:7

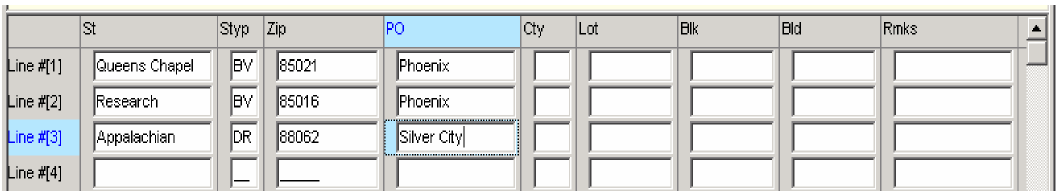
	Ln#	Del	St	Styp	Zip	PO
Line #[1]	1		Queens Chapel	BV	85021	Phoenix
Line #[2]	2		Research	BV	85016	Phoenix
Line #[3]	3		Appalachian	DR	20770	Greenbelt
Line #[4]	4		Appalachian	DR	85016	Phoenix
Line #[5]	5					
Line #[6]	6					
Line #[7]						
Line #[8]						
Line #[9]						



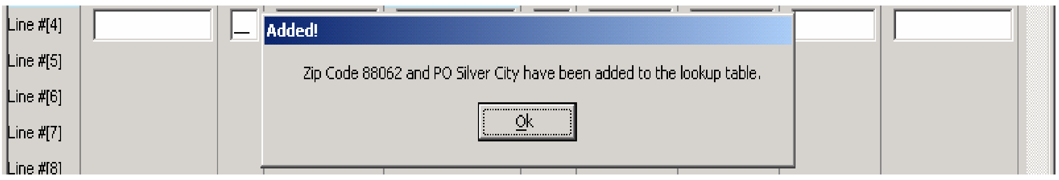
	Ln#	Del	Pmt#	Mo	Day	HUs	Hse#	St	Styp	Zip	PO
Line #[1]	1		3641	2	2	3	6161	Queens Chapel	BV	85021	Phoenix
Line #[2]	2		3642	2	4	1	4820	Research	BV	85016	Phoenix
Line #[3]	3		3643	2	5	1	11830	Appalachian	DR	20770	Greenbelt
Line #[4]	4		3644	2	6	10	371	Appalachian	DR	85016	Phoenix
Line #[5]	5		3645	2	7	1	4081	Queens Chapel	BV		
Line #[6]	6										

4.6. Adding Data to the Lookup

When the FR runs the Maniplus script (by pressing "Ctrl +A", for Add in our case) to add data to the lookup table, the data inside the fields that are selected are captured and put into auxfields in the script. The data is copied from the auxfields to the lookup table, and it is written to the table. We have also set up a DIALOGBOX to pop up and tell the FR that the information was added and what the information was.



	St	Styp	Zip	PO	Cty	Lot	Blk	Bld	Rmks
Line #[1]	Queens Chapel	BV	85021	Phoenix					
Line #[2]	Research	BV	85016	Phoenix					
Line #[3]	Appalachian	DR	88062	Silver City					
Line #[4]									



Line #[4]
Line #[5]
Line #[6]
Line #[7]
Line #[8]

Added!

Zip Code 88062 and PO Silver City have been added to the lookup table.

4.7. Accessing the Maniplus Scripts

Blaise 4.7 now allows Maniplus to be called as part of the actions available from the DEP Menu (both menu items and button bar buttons) and the Action Dropdown in the Datamodel Properties. Since the actions dialog calls Maniplus via a command line, we can use parameters to control how the script is run. This has given us the ability to have the code that reads data from the lookup table and the code that adds data to the lookup table reside inside the same script. In our instruments, we have set the parameter of 'L' to perform the lookup and 'A' to add to the lookup table. To go even further, Blaise can look at the name of the field that the FR is on when the script is called (based on ACTIVEFIELD) and read from or write to the appropriate table. This is beneficial in that the same function key combination (Ctrl + L for us) can be used to call different dynamic lookup tables since only one script is needed to access all of the tables. A standard function key makes this functionality easier on the FRs.

See Example 2 in the Appendix for an example of a Maniplus script with parameters controlling the flow of data

5. Toggle Between "Line View" and "Full View"

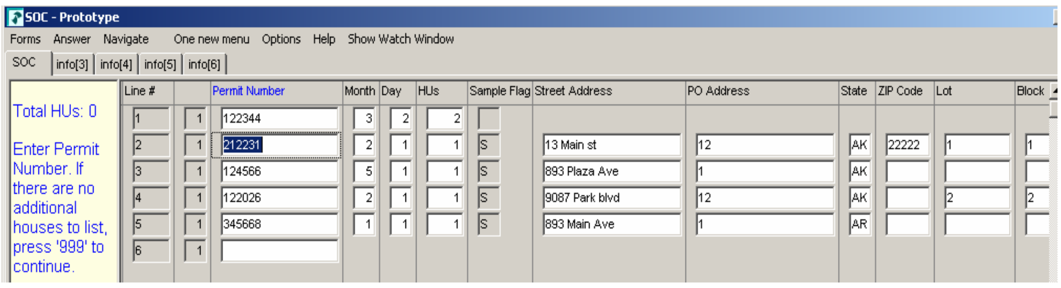
Due to the number of variables displayed/collected in the row of a table (about 25 variable per row), we have been required to allow for a "Full View" display/collection screen. The Sponsor requested that this option be a "Toggle" that goes from "Line View" to "Full View" and vice versa. This option should allow the FR to collect data in both views and toggle at any time.

5.1. Initial Approach – Setting Block Row Inside Table as Parallel

The toggle requirement can be achieved by setting the block row inside the table as parallel. When we initially tried this inside the Listing instrument we observed some degradation in the performance from field to field. This was mainly because we were setting a large number of rows (two thousand rows) as parallel. Additionally, there was no way to associate each tab to one function key so it would be very hard to use this approach.

```
PARALLEL
Info=Listing2.Blisting2
```

Where Info is set as parallel and where Blisting2 is an arrayed block up to 2,000 rows. Since a normal listing operation ranges from 800 to 1500 permits, the amount of tabs would be hard to visualize and cumbersome to the FR. See the graphic below.


SOC - Prototype
Forms Answer Navigate One new menu Options Help Show Watch Window
SOC info[3] info[4] info[5] info[6]

Total HUs: 0

Enter Permit Number. If there are no additional houses to list, press '999' to continue.

Line #		Permit Number	Month	Day	HUs	Sample Flag	Street Address	PO Address	State	ZIP Code	Lot	Block
1	1	122344	3	2	2							
2	1	212231	2	1	1	S	13 Main st	12	AK	22222	1	1
3	1	124566	5	1	1	S	893 Plaza Ave	1	AK			
4	1	122026	2	1	1	S	9087 Park blvd	12	AK		2	2
5	1	345668	1	1	1	S	893 Main Ave	1	AR			
6	1											

5.2. Better Approach – Use Maniplus

After consultation with Statistics Netherlands, the approach we decided to use to implement the "toggle" functionality in a more robust manner was to use Maniplus.

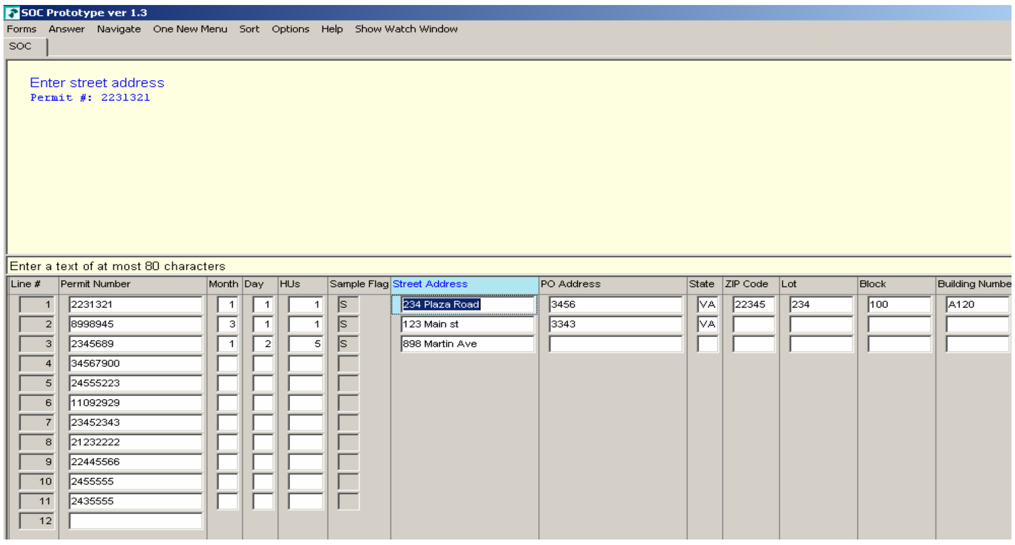
The steps to do this are as follows:

1. Create an instance of the block used in the table and set this new Block as parallel.

2. Create a Function Key (button) and associate it with an event.

3. Have this "event" call a Maniplus script that would capture the data of the arrayed block from which it was invoked (its instance or line row) and transfer to the parallel block defined. Then set the focus to this parallel via the Maniplus script.

4. In order to return to the main path - either by using a function key, button, or at the end of the parallel block - another event would take place using Maniplus script sending data from the current parallel block to the arrayed block.

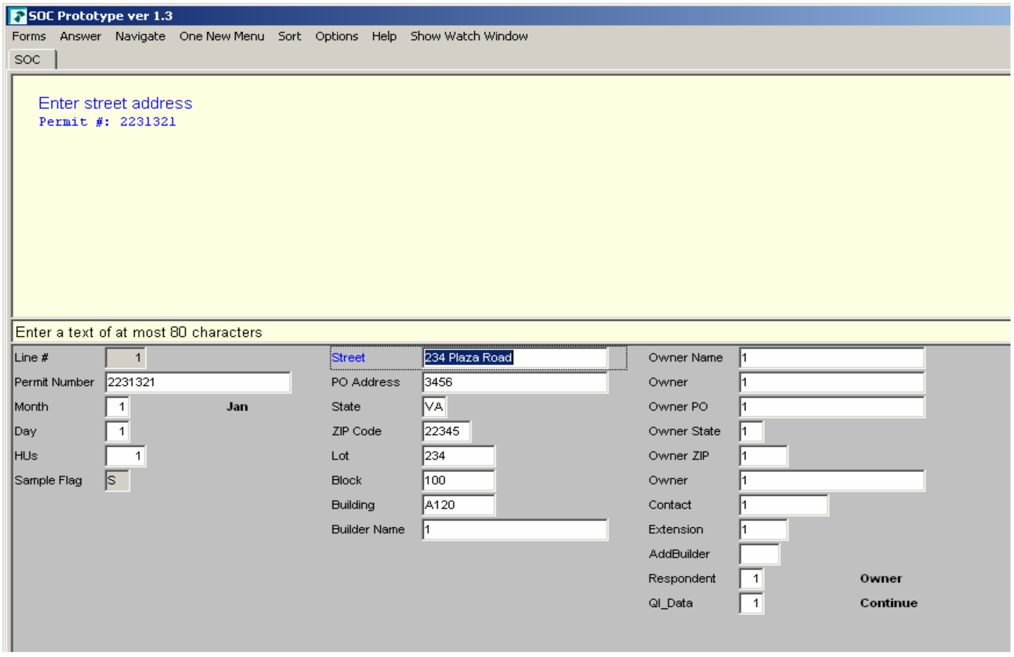

SOC Prototype ver 1.3
Forms Answer Navigate One New Menu Sort Options Help Show Watch Window
SOC

Enter street address
Permit #: 2231321

Enter a text of at most 80 characters

Line #	Permit Number	Month	Day	HUs	Sample Flag	Street Address	PO Address	State	ZIP Code	Lot	Block	Building Numbe
1	2231321	1	1	1	S	234 Plaza Road	3456	VA	22345	234	100	A120
2	8998945	3	1	1	S	123 Main st	3343	VA				
3	2345689	1	2	5	S	898 Martin Ave						
4	34567900											
5	24555223											
6	11092929											
7	23452343											
8	21232222											
9	22445566											
10	2455555											
11	2435555											
12												

Example of the "Full View" After Executing the Toggle:


SOC Prototype ver 1.3
Forms Answer Navigate One New Menu Sort Options Help Show Watch Window
SOC

Enter street address
Permit #: 2231321

Enter a text of at most 80 characters

Line #	1		Street	234 Plaza Road	Owner Name	1
Permit Number	2231321		PO Address	3456	Owner	1
Month	1	Jan	State	VA	Owner PO	1
Day	1		ZIP Code	22345	Owner State	1
HUs	1		Lot	234	Owner ZIP	1
Sample Flag	S		Block	100	Owner	1
			Building	A120	Contact	1
			Builder Name	1	Extension	1
					AddBuilder	
					Respondent	1 Owner
					QI_Data	1 Continue

See **Example 4** of the Appendix for an example of code that shows the "Line View" to "Full View" switch.

6. Using Different Listing Tables Based on FR Preference

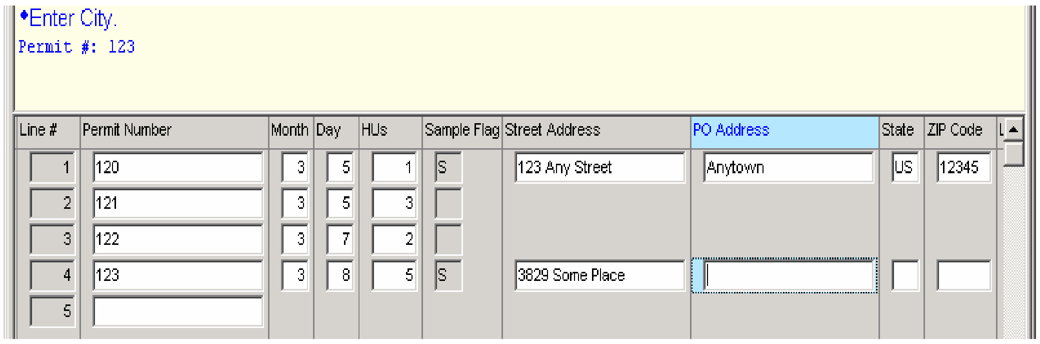
In the SOC Listing Instrument, not all permits that are listed are going to fall into a final sample that requires "additional information" on the permit to be collected. As discussed earlier, the sponsor has a requirement to allow the FR to choose how they wish to collect the permit information:

1. Collect minimal information for every permit first (Permit Number, Month/Day Issue, and # of HUs) and then collect "additional information" after "final" sampling has been run for only those permits that fall into final sample.

2. Collect all information needed (including additional information) for permits that may fall into final sample while they have the permit in hand. Once "final" sample is run, some of those permits will fall out of sample.

Through the use of flags, we can set the behavior of the listing table to collect the full permit information now, or to collect the information after we run our final sampling. In the beginning of the interview, the FR is asked if they want to collect additional permit information while they are listing or after they are done listing.

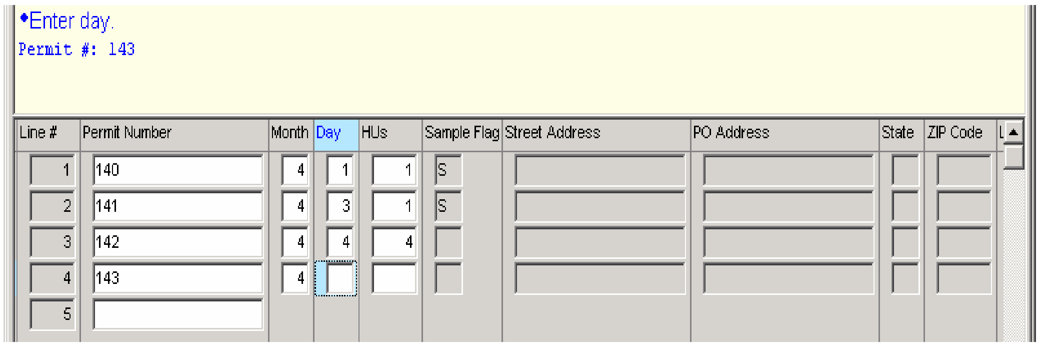
If they decide to collect the additional information during listing, the table will collect the permit number, permit month, permit day, and number of housing units for each permit line. If the permit falls into a potential sample, additional information questions are brought on path to be answered by the FR.


•Enter City.
Permit #: 123

Line #	Permit Number	Month	Day	HUs	Sample Flag	Street Address	PO Address	State	ZIP Code	L
1	120	3	5	1	S	123 Any Street	Anytown	US	12345	
2	121	3	5	3						
3	122	3	7	2						
4	123	3	8	5	S	3829 Some Place				
5										

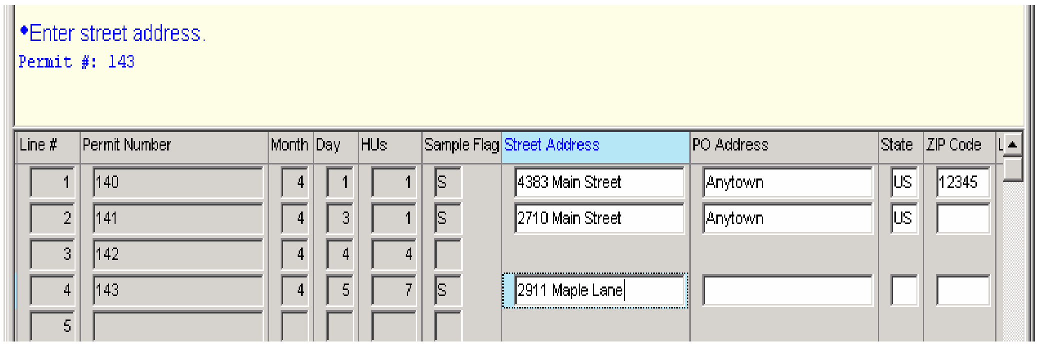
After listing is complete, the final sampling is run and since the additional information is already collected, the user can continue moving through the survey.

If the user opts to collect additional information after completing the listing, they will enter the permit number, permit month, permit day, and number of housing units for each line, but no additional information will be collected until after the final sampling is run.


•Enter day.
Permit #: 143

Line #	Permit Number	Month	Day	HUs	Sample Flag	Street Address	PO Address	State	ZIP Code	L
1	140	4	1	1	S					
2	141	4	3	1	S					
3	142	4	4	4						
4	143	4								
5										

When the user leaves the listing table they are asked if the listing is complete. If they answer "no", they would return to the listing table. If they enter "yes", the final sampling is run. After final sampling is run, the additional information questions are brought on path.



•Enter street address.
Permit #: 143

Line #	Permit Number	Month	Day	HUs	Sample Flag	Street Address	PO Address	State	ZIP Code	L
1	140	4	1	1	S	4383 Main Street	Anytown	US	12345	
2	141	4	3	1	S	2710 Main Street	Anytown	US		
3	142	4	4	4						
4	143	4	5	7	S	2911 Maple Lane				
5										

After entering the additional information, the FR will move on through the rest of the survey.

The code to accomplish this process needs one flag to help set the path; the rest of the pathing is based on answers given by the FR.

See **Example 3** in Appendix for an example of how to accomplish this task.

7. Controlling When to Pop-up a Lookup Table

7.1. Why We Want to Control This

Normally, if an external lookup table is used in the instrument, the lookup is set to pop up when the FR begins entering data into the field. Typically, these lookup lists are only called one time in an instrument so it makes sense to have it automatically pop up.

For listing instruments, however, this approach does not make as much sense. It would mean that the lookup list would pop up for every line listed. For example, if a street type lookup list is used and the FR listed 400 lines of data, the FR would see this list pop 400 times. Since the FRs conduct many listing assignments, they will most likely already know most of the common street type abbreviations to use. But, for times when there is an unusual street type, the sponsor would like the FRs to have the ability to select that street type from a list.

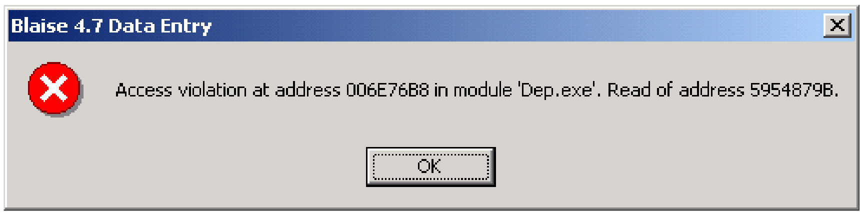
7.2. The Approach

Our first approach to controlling how the lookup table behaves was to turn on the “Use Normal Asker” option in the ModeLib editor and then set a custom startup behavior for the asker in that field. In our case we used the pressing of the “Insert” key to pop up the lookup table. This meant that fields using a lookup had to be set up as a type in order to show up in the Datamodel properties. This approach worked until a problem developed with using external lookup tables in the rules.

Our workaround for this problem was to call the lookup tables from Maniplus and use a DIALOGBOX, but that meant that Maniplus needed to be called from within the instrument. This was accomplished by using a menu item that was enabled only when the cursor was on a certain field ($FIELDTAG = ‘FieldTagName’). We went further and created an action button in the field as well. The usage of this button worked okay, but it would force our FRs to use a mouse to click on the button. The issue here is that we don’t like to require the FRs to use the mouse pad to conduct interviewing so we ended up taking the button out and sticking with the menu item.

7.3. Memory Access Error

During the course of this research, a problem developed in Blaise that forced us to change our approach to using external lookup tables. Normally we define the EXTERNALS to be Blaise databases and only use the filename when it is needed. However, in Blaise 4.7.1 b989 and Blaise 4.7.1 b1000 certain Maniplus scripts would not run in the DEP session. It occurred with Maniplus scripts that used the INTERCHANGE setting to access the data in the DEP session. If we tried to run one of the Maniplus scripts while in the DEP session, the following error would occur:



Access violation at address 006E76B8 in module 'Dep.exe'. Read of address 5954879B.

If we were running Maniplus in quiet mode, this error caused our instrument to lock up because there was no known way to stop Maniplus from continuing to run. To get around this problem, we changed the setting for the external tables from BLAISE to MEMORY. It fixed the problem with the access violations. However, this solution presents us with another problem. That is if we used multiple lookup tables inside the rules, we will have to load each one into memory. This means the instrument needs to take up more memory, and it could affect performance.

Ultimately we removed the offending lookup table because it was a static table of abbreviations for street types (DR, ST, RD, etc.) that an experienced FR would be familiar with. If the FR did not know a certain street type abbreviation, we created an item-level help screen for them that they could reference. We also added edits to this field to verify a valid street type was entered.

8. Conclusion

Blaise 4.7 has been extremely helpful in allowing us to address many of the different and complex requirements requested by our sponsors. Without this version of Blaise it would be extremely difficult, if not impossible, to address some of the requirements. Since our various prototypes work with Blaise 4.7, we feel pretty confident that it will allow us to give our sponsors most of the listing requirements that they have requested.

9. Acknowledgements

We would like to acknowledge the following people for input into this paper and for input into how to address some of the requirements requested:

Statistics Netherlands
Karen Bagwell, U.S. Census Bureau
Tom Spaulding, U.S. Census Bureau

Appendix: Code Examples

Example 1 – Creating the Lookup Table

```
DATAMODEL ZipAndPO
   SECONDARY
       ZipFirst = ZipCode, PO
       POFirst = PO, ZipCode
   FIELDS
      ZipCode "Enter the Zip Code" / "ZipCode" : STRING[5], EMPTY
      PO "Enter the PO Name" / "PO" : STRING[28], EMPTY
   RULES
      ZipCode
      PO
ENDMODEL
```

Example 2 – Maniplus Script with Parameters Controlling the Flow of Data

```
PROCESS LookupTown
   USES
      inst                  {Datamodel of the DEP}
      ZipAndPO              {Datamodel of the Zip Code and PO lookup table}

UPDATEFILE          {Used to read in and write to the lookup table}
      UpdateFile1 : ZipAndPO('ZipAndPO.bdb', BLAISE)
         SETTINGS
            OPEN = YES
            CONNECT = NO
            MAKENEWFILE = NO  {Append the current file}
TEMPORARYFILE
      DEP : inst                             {File used to link to the DEP}
         SETTINGS
            INTERCHANGE = TRANSIT       {Used to link Manipula to the current DEP session}
AUXFIELDS
      LineNum_in : INTEGER    {The line number the instrument is on when this script is called}
      NumString : STRING[25]   {Used to collect the current line number from ACTIVEFIELD}
      Button_Select : INTEGER  {Button for the dialogbox}
      Button_Ok : INTEGER
      Param_In : STRING[1]      {Holds the parameter used when this script is called}
      ZipCode : STRING[5]
      PO : STRING[28]
DIALOGBOX TownLookup  {A dialog box with the Zip and PO lookup table}
      ESCAPE = NO
      LOOKUP UpdateFile1
         SEARCH = YES
      BUTTON Button_Select
         CAPTION = '&Select'
         VALUE = 1
      BUTTON Button_Select
         CAPTION = '&Cancel'
         VALUE = 0
DIALOGBOX ZipAdded
      SIZE = (400,75)
      ESCAPE = NO
      TEXT = ('Zip Code ^ZipCode and PO ^PO have been added to the lookup table.',30,10)
      BUTTON Button_Ok
         CAPTION = '&Ok'

MANIPULATE
NumString                                                                                    :=
SUBSTRING(ACTIVEFIELD,POSITION('[',ACTIVEFIELD)+1,(POSITION(']',ACTIVEFIELD)-
POSITION('[',ACTIVEFIELD)-1))
      LineNum_in := VAL(NumString)
      Param_In := parameter(1) {Get the parameter that was used when the script was called}

      IF Param_In = 'L' THEN {Use the lookup table}
```

```
            TownLookup('Select Town/Zip Code')
            IF Button_Select = 1 THEN {Zip and PO Selected}
               UpdateFile1.READ   {Take the record from the table and fill the appropriate DEP fields}
               Listing.BPALListing[LineNum_in].ZipCode := UpdateFile1.ZipCode
               Listing.BPALListing[LineNum_in].PO := UpdateFile1.PO
            ENDIF {Button_Select = 1}
      ENDIF {Param_In = 'L'}

      IF Param_In = 'A' THEN {Add the Zip and PO from the current line to the lookup table}
         ZipCode := Listing.BPALListing[LineNum_in].ZipCode
         PO := Listing.BPALListing[LineNum_in].PO
         UpdateFile1.ZipCode := ZipCode
         UpdateFile1.PO := PO
         UpdateFile1.WRITE
         ZipAdded('Added!')
     ENDIF {Param_In = 'A'}
```

Example 3 – Code for Controlling Different Listing Tables

```
BLOCK blkListing
   IMPORT in_LineNum : INTEGER
   IMPORT in_CollectInfo : TaddInfo
   IMPORT in_ListingFlag : TYESNO

 FIELDS
     LineNum (LineNum) / "Line #" : 1..3000   {Var to hold the location of the permit in the table}
     PermitNum (PermitNum) "Enter Permit Number, if there are no additional houses to list, press
     '999' to continue /    "Permit Number" : STRING[24]
      HUs (HUs) "Enter the number of housing units on the permit." / "HUs" : 1..1500
      SampleFlag (SampleFlag) / "Sample Flag" : STRING[1]
      Address (Address) "Enter the street address" / "Address" : STRING[80]
       …

   RULES
      LineNum := in_LineNum            {Set line number based on array index}

      IF in_ListingFlag = Yes THEN  {FR collects info after listing & indicate listing is done..}
         PermitNum.SHOW              {…then show the permit number, date of permit, HU's, etc}
         SampleFlag.SHOW
         …
      ELSE           {Required information if listing is not done, or FR opts to collect everything now}
         PermitNum
         HUs
         IF HUs = 1 OR HUs >= 5 THEN   {Condition for setting potential sample}
            SampleFlag := 'S'
         ELSE
           SampleFlag := EMPTY
         …
         ENDIF
      ENDIF {in_ListingFlag = Yes}

      IF PermitNum = RESPONSE THEN  {Bring the rest of the listing line on path when   answered}
         IF in_CollectInfo = AfterList AND (in_ListingFlag = No or in_ListingFlag = EMPTY) THEN
           Address.SHOW                              {User opts to collect info after listing}
            …
         ELSEIF (in_CollectInfo = WhileList OR in_CollectInfo = AfterList) AND SampleFlag = 'S'
         THEN
            Address {Additional information to be collected because listing is done or user opts to
            collect all info while listing}
                    …
         ENDIF
      ENDIF
ENDBLOCK



TABLE tblListing
```

```
TYPE
      tAddInfo (WhileList "While Listing", AfterList "After listing is complete")
LOCALS
      I : INTEGER[4]    {Counter to set position of the table.
FIELDS
      Listing : ARRAY[1..3000] of blkListing
      CollectInfo (CollectInfo) "Would you like to collect additional info during listing or after listing
           is complete?" /  "CollectInfo" : tAddInfo
      ListingFlag :  tYesNo
RULES
      ListingFlag.keep
      CollectInfo                          {Ask user how they want to collect permit info}
      FOR I := 1 TO 3000 DO
         IF I = 1 OR (BListing2[I-1].PermitNum <> EMPTY)THEN
             BListing2[I](I, CollectInfo, ListingFlag)       {Go into the block and add information}
         ENDIF
      ENDDO
ENDTABLE
```

Example 4 – Switching from "Line View" to "Full View"

PROCESS SwitchParallel   (**Manipula Code)**

```
AUXFIELDS
  ParallelName: STRING
  i: INTEGER
 PROCEDURE prc_Parallel     {find out to which parallel the active field belongs...}
  i:= position('.',ParallelName,position('.',ParallelName)+1)
  IF I>0 THEN
    ParallelName:= substring(ParallelName,1,i-1)
  ENDIF
  IF uppercase(ActiveParallel)=uppercase(ParallelName) then
    setactiveparallel('ParallelJumpDemo') {back to main}
  ELSE
    Setactiveparallel(ParallelName)
    setactivefield('')            {clear active field = go to first field of active parallel }
 ENDIF
ENDPROCEDURE

MANIPULATE
  ParallelName:= activefield
  prc_Parallel
```

Blaise Code Example

```
DATAMODEL ParallelJumpDemo
PARALLEL
   ATable.Rows
BLOCK TABlock
FIELDS
   field1, field2, field3: STRING[2], EMPTY
ENDBLOCK

TABLE TATable
 FIELDS
   rows:ARRAY[1..10] OF TABlock
ENDTABLE
FIELDS
  Q : String[4], EMPTY
  ATable: TATable
ENDMODEL
```