DEP Unchained: Using Manipula on the Current Record in Memory

Peter Stegehuis and Rick Dulaney, Westat, Inc

INTRODUCTION

One longstanding feature of Blaise is the very tight integration between code, presentation and data storage: what you code is what you see, and what you see is what's stored in the database. The descriptive nature of Blaise datamodels also provides a framework for the Blaise selective checking mechanism, which is generally regarded as one of the strengths of the Blaise system.

However, these advantages come with two significant potential pitfalls. First is that Blaise datamodels require fixed arrays. This requirement is fine to a point, and accommodates many surveys, but can be problematic when arrays are nested or combined. Take the simple example in Listing 1, in which a list of persons are each asked about a list of cars:

Listing 1. Collecting Data about Persons and their Cars

```
DATAMODEL CarPeople

  BLOCK BCars
    FIELDS
      MakeModel "What's the make and the model of the car?": STRING[20], EMPTY
      LikeIt "Did you like this car": (Yes, No, SoSo)
    RULES
      MakeModel
      IF MakeModel <> EMPTY THEN
        LikeIt
      ENDIF
  ENDBLOCK {BCars}

  BLOCK BPerson
    LOCALS
      J: INTEGER

    FIELDS
      FirstName "First Name": STRING[20], EMPTY
      LastName  "Last Name" : STRING[30]
      Cars: ARRAY[1..500] OF BCars

    RULES
      FirstName
      IF FirstName <> EMPTY THEN
        LastName
        FOR J:= 1 TO 500 DO
          IF J=1 OR Cars[J-1].MakeModel <> EMPTY THEN
            Cars[J]
          ENDIF
        ENDDO
      ENDIF
  ENDBLOCK {BPerson}

  LOCALS
    I: INTEGER

  FIELDS
    Person: ARRAY[1..500] OF BPerson

  RULES
    FOR I:=1 TO 500 DO
      IF I = 1 OR Person[I-1].FirstName <> EMPTY THEN
        Person[I]
      ENDIF
    ENDDO
ENDMODEL {CarPeople}
```

When prepared, this datamodel will yield an error ("The maximum number of allowed pages has been reached"), and the normal course of action is to compromise on the size of the arrays until it prepares successfully. This is just one example, created to trigger this error, but we regularly run into similar

problems in large establishment surveys with hundreds of observations nested at multiple levels, and even in complex household studies, particularly those that track populations and rostered information over time.

The second potential pitfall is the datamodel-wide scope of the selective checking mechanism itself, which checks the entire datamodel at each opportunity, including checks and downstream computations that may not be intended. When the interviewer leaves the first field in a ten-item datamodel, Blaise checks and potentially performs computations after the tenth item as well. This leads to awkward IF…THEN blocks to "protect" code from running before the programmer intended it, and may lead to dangerous uses of the powerful KEEP statement.

NEW CAPABILITIES

Blaise 4.8 introduced a powerful new capability: the Blaise DEP may shell out to other programs, which can perform additional computations, and even write directly to the record in memory from outside the DEP. Because Manipula has so many broad uses within the Blaise system, we focus in this paper on the use of external Manipula procedures running outside the DEP. However, any executable recognized on your system will run in the same fashion.

External programs may be associated with several different actions in the DEP, such as a button press or a category selection. Buttons on a panel may become visible or active based on conditional logic using the interview data. Figure 1 shows one simple method: open Project->Datamodel Properties, select the type, and click the Event button to specify the program and procedure name – in this case, we will run the Manipula setup CarPeople and start at the procedure named StartCarsProc.

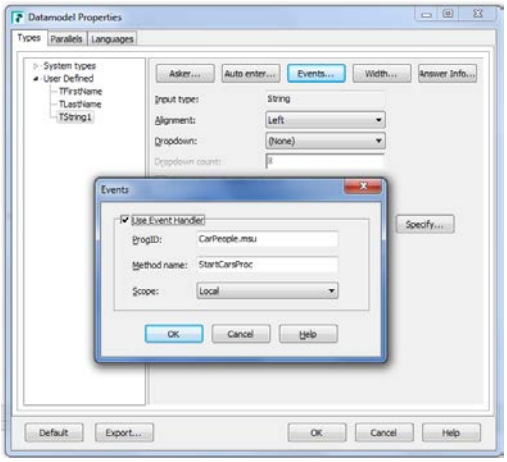

Figure 1. Associating an external program with a field type.

DISTRIBUTED DATAMODELS

One of the immediate impacts of this new capability is the ability to accommodate large datamodels, in a way that resembles traditional relational database operations. Returning to the example above, the natural way to model this data is to have one table for persons having a one-to-many relationship with a table storing information about each car that a person owns. We can now collect store these data using two datamodels and use Manipula to move between them. The People datamodel has a table with one row per person:

Listing 2: People datamodel

```
DATAMODEL People
  PRIMARY
    CaseNumber

  TYPE
    TFirstName = STRING[20]
    TLastName  = STRING[30]
    TString1   = STRING[1], EMPTY

  FIELDS
    CaseNumber: 1..1000
    CurrentPersonID: 1..500

  TABLE BAllPeople
    LOCALS
      I: INTEGER

    BLOCK BPerson
      FIELDS
        PersonID: 1..500
        FirstName "First Name": TFirstName, EMPTY
        LastName  "Last Name" : TLastName
        AddCars   "Press Enter to add cars for ^FirstName ^LastName ": TString1, EMPTY

      RULES
        PersonID:= I
        PersonID.SHOW
        FirstName
        IF FirstName <> EMPTY THEN
          LastName
          AddCars
        ENDIF
    ENDBLOCK {BPerson}

    FIELDS
      Person: ARRAY[1..500] OF BPerson

    RULES
      FOR I:=1 TO 500 DO
        IF I = 1 OR Person[I-1].FirstName <> EMPTY THEN
          CurrentPersonID:= I
          Person[I]
        ENDIF
      ENDDO
  ENDTABLE {BAllPeople}

  FIELDS
    AllPeople: BAllPeople
```

The AddCars column in the person table uses the TString1 format above to call Manipula (recall that the method name is StartCarsProc):

Listing 3. CarPeople Maniplus Setup.

```
PROCESS CarPeople

USES
  People
  Cars

UPDATEFILE
  CarsData: Cars ('Cars', BLAISE)

TEMPORARYFILE PeopleInMemory: People
  SETTINGS
```

```
    INTERCHANGE = SHARED
    CONNECT = NO
    CHECKRULES = YES

AUXFIELDS
  Rslt: INTEGER
  AuxStrPeopleCaseID, AuxStrCurrPersID: STRING[4]

PROCEDURE StartCarsProc
  IF (ROUTERSTATUS= BLRSPOSTEDIT) THEN
    AuxStrPeopleCaseID:= STR(PeopleInMemory.CaseNumber)
    AuxStrCurrPersID:= STR(PeopleInMemory.CurrentPersonID-1)
    Rslt:= CarsData.EDIT('/G /K'+ AuxStrPeopleCaseID + ';' + AuxStrCurrPersID + ' /X')
  ENDIF
ENDPROCEDURE {StartCarsProc}
```

Note that the current record in memory is designated PeopleInMemory and uses the INTERCHANGE = SHARED keyword to enable writing from Manipula and subsequent datamodels. The one-to-many relationship between People and Cars is implemented by constructing the Cars key using the case ID from People plus the ID of the current person and calling the Cars datamodel to collect make/model and consumer attitude:

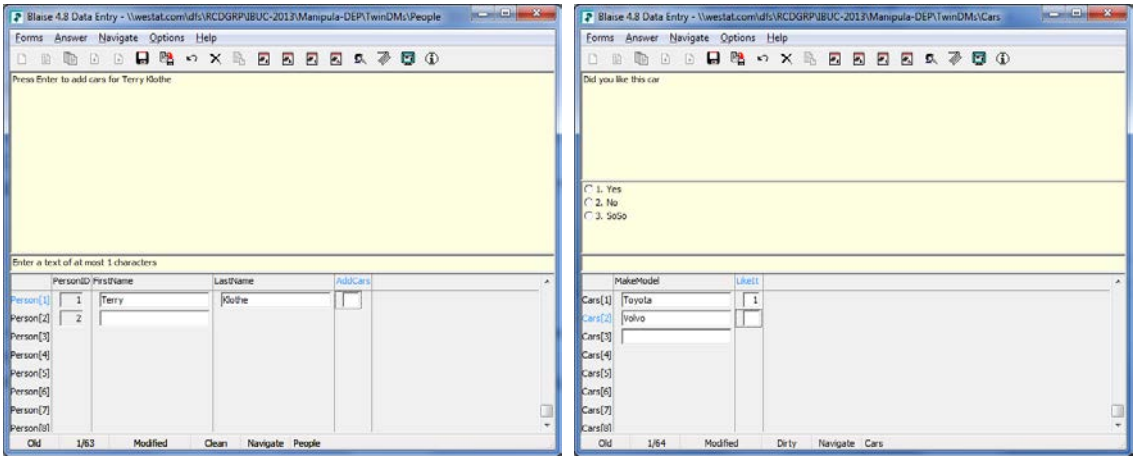

Figure 2. Calling the Cars datamodel while the People datamodel is active.

Note in the title bar on the left that the People datamodel is running to collect a table of persons. Pressing enter in the AddCars column goes through Manipula and presents the Cars datamodel on the right. Pressing Enter in on a blank line will return to the People datamodel to collect the next person and his or her Cars. For this example, we preloaded records in the databases, but this is normally implemented using one or more foreign keys. Many-to-many relationships may use a separate datamodel to store the linkages and other attributes of the relationship. There is no logical limit to the number of datamodels or Manipula setups that may be invoked.

The ability to extend Blaise capabilities at run-time offers significant advantages. Of course, once Manipula has started, one can show dialogs, open additional datamodels, or perform computations involving any number of databases – including the one currently opened by the original datamodel. Of particular importance is that Manipula code called in this way can write directly to the DEP record in memory. This has significant impact for large or complex surveys that threaten to exceed the storage capabilities of a single datamodel, or have complex navigation requirements. Some examples of activities that may run while the DEP is active include:

- Sampling units for subsequent interviewing in the DEP
- Various lookup activities, with outcomes stored in the current record in memory
- Roster management for lists currently in use in the DEP, including add-delete-update operations

NAVIGATION

Navigation generally poses challenges in large instruments. To facilitate non-linear navigation, such as moving back to a specific field, developers often resort to techniques such as emptying a field on or near the destination. The next time the selective checking mechanism fires, the interviewer will be taken back to that now-empty field. This solution requires that the field not have the EMPTY attribute and that the programmer be very careful to avoid unintended navigation. In Blaise 4.8 developers may use the SETACTIVEFIELD keyword to send the interviewer to any location in the current record, regardless of whether the field is required or whether there is indeed data in the field.

This capability is extraordinarily powerful when combined with the ability to distribute data among different datamodels. In the simple example above, at the end of adding cars for one person, you automatically return to the People datamodel, because that's where you left off. In a more elaborate example you could more actively determine where to 'put the cursor' by using the keyword SETACTIVEFIELD to return control to the original datamodel and continue data collection using the DEP. ACTIVEFIELD is the corresponding property, so programmers can save the active field at any point and return to it later on. The related keywords ACTIVEPARALLEL and SETACTIVEPARALLEL are also very useful; one can interrupt the current processing using a parallel block, shell out to Manipula to perform other activities, and then return to the parallel block or to the original datamodel.

CASE STUDY

There are many potential uses for these capabilities. We show here a specific example of a last-minute requirement that would have been difficult or impossible to implement within a single datamodel.

The National Hospital Care Study is a large and complex study, one portion of which requires data collectors to abstract medical records from hospitals. The data collection begins with a hospital level induction instrument, which collects general information and enumerates ambulatory units (AUs). The system then creates a case for each AU and the data collector opens the AU instrument. The AU instrument samples patient visits to that AU during a reference period and collects data onto a Patient Record Form for that visit; the PRF has many individual parts, as shown here:

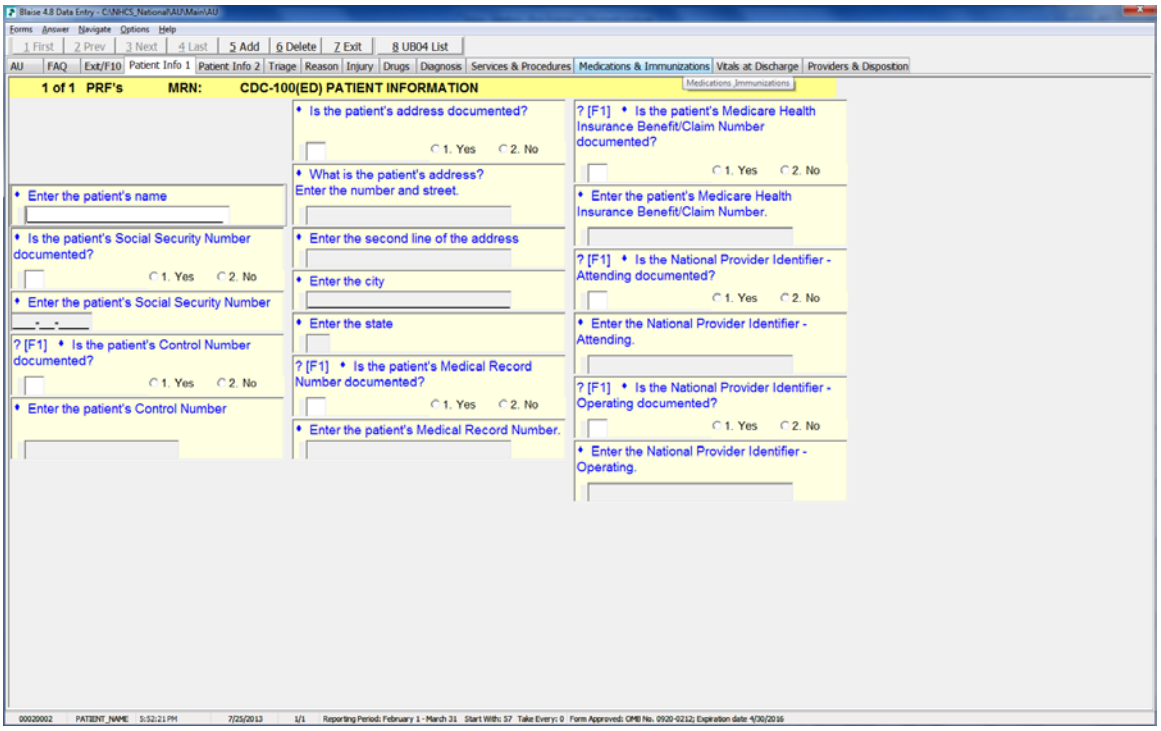

Figure 3. The main Patient Record Form screen in the AU instrument.

In Figure 3, a single PRF is collected using multiple tabs. Data collectors may use buttons 1-7 on the panel to move through collected PRFs as well as to add, delete or edit a PRF.

In order to increase the number of cases meeting a specific condition, late in the development cycle the client provided sample specifications that allowed us to create an external file of hospital visit data as reported on a specific claims form called the UB04. Unfortunately, the external UB04 data was at the hospital level, not the AU level. We added a button – number 8, above – to enable interviewers to examine the external file, sort it as necessary, add comments or a disposition if appropriate, and select a UB04 which is then written back to the current PRF record in memory.

Figure 4 shows the screen after pressing the "8 – UB04 List" button. We have called a Manipula setup which reads in the external data and presents it to the data collector. Notice that the first record displays a disposition ("Incomp"), and the data collector is entering a comment on the second record. When the data collector then pressed the "Move Case to PRF" button on the third record, the data were written to the PRF record as shown in Figure 5.

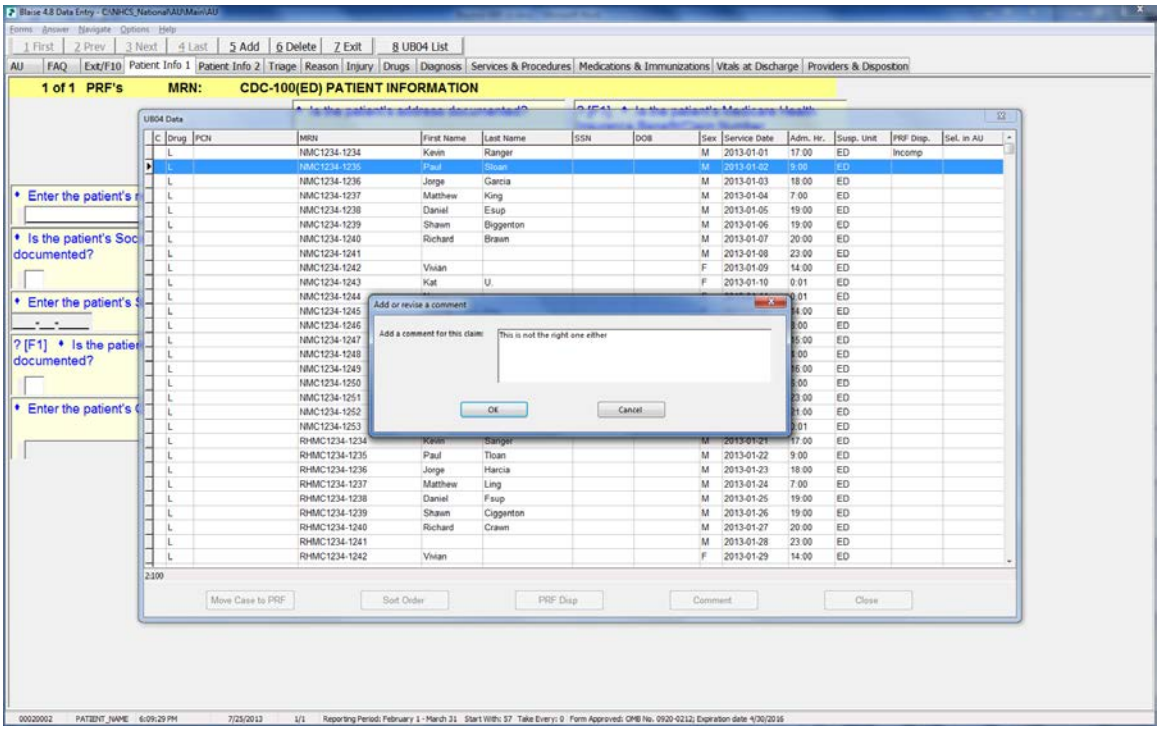

Figure 4. Updating disposition and adding comments in the external file.

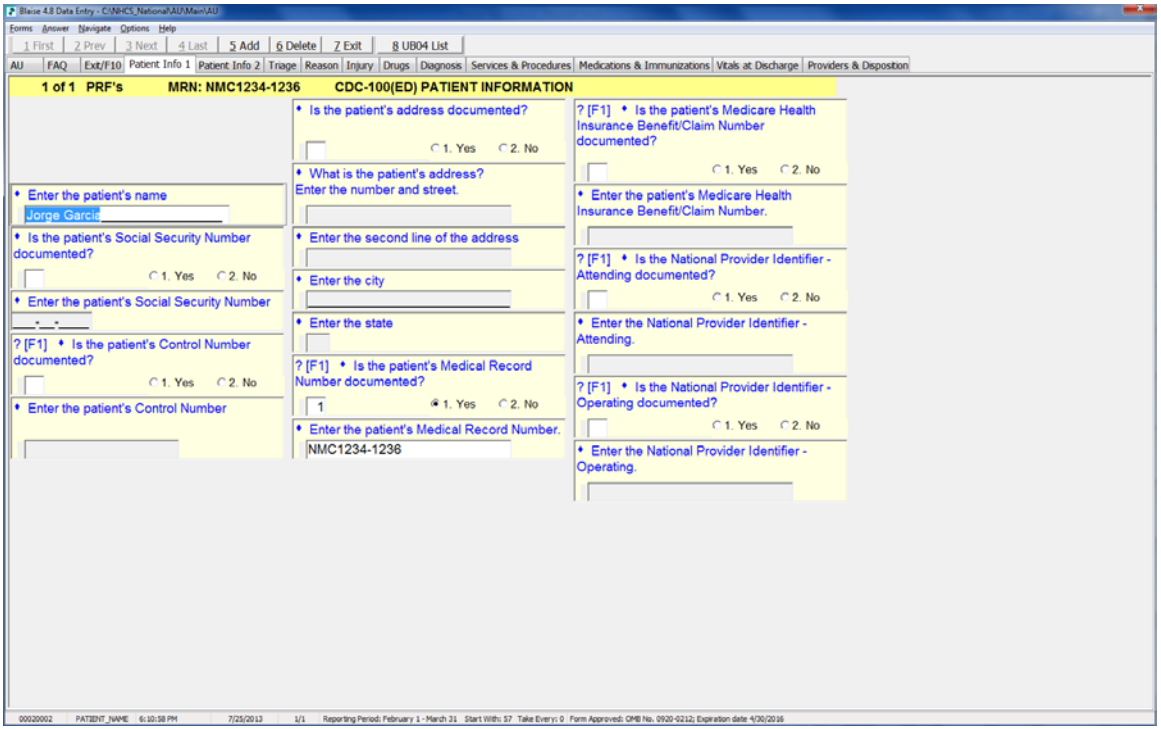

Figure 5. Data from external file written back to DEP record in memory.

In this case, when we copy UB04 data to the Blaise record in memory, we also indicate this on the external file, as shown by the Blaise case number in the rightmost column in Figure 6.

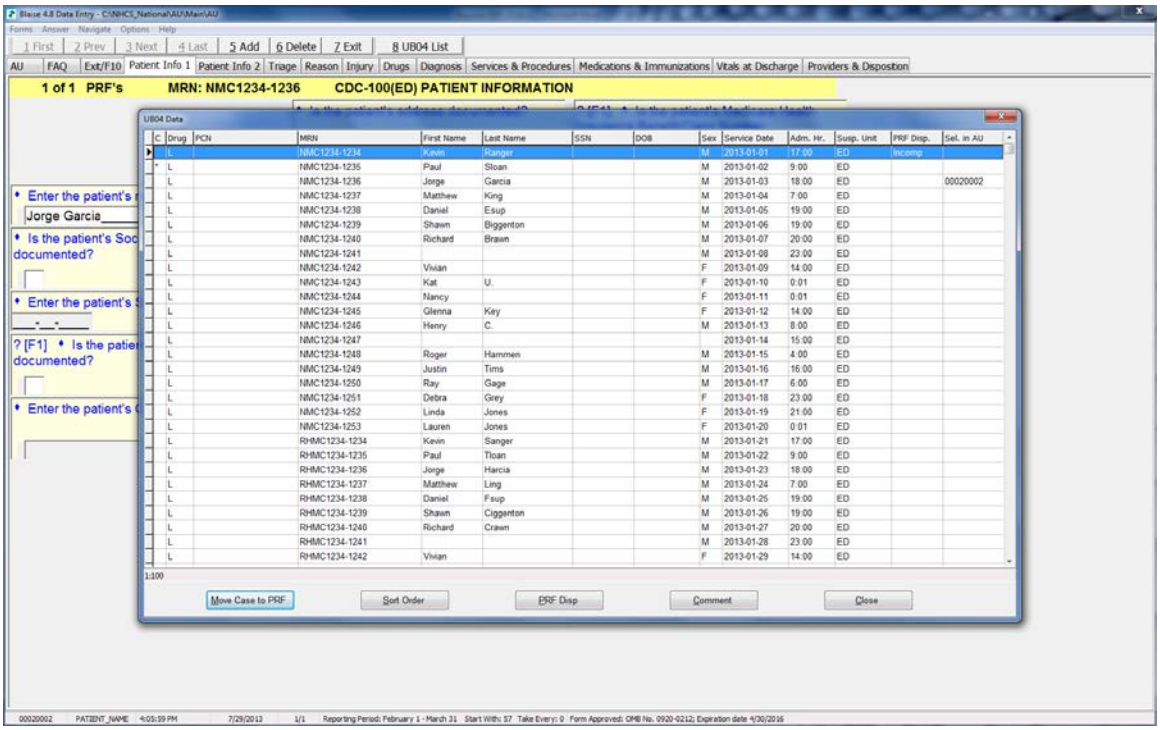

Figure 6. Storing the Blaise case ID on the external file.

The details of the NHCS are considerably more complex, and attempting to build this capability purely within the DEP would have been extremely challenging. The techniques added in Blaise 4.8 and described here allowed us to implement this solution within the project schedule.

SUMMARY

Blaise 4.8 allows developers to move within and between datamodels during data collection, organizing data into entities most appropriate for their projects. The full range of Manipula functionality is available at run-time during DEP. This movement can be interviewer-initiated through use of a button or question response, or can be initiated through code. The power of this technique has broad application across our most complex surveys.