

# Some different uses of expressions with Blaise 5 resource databases

Arthur Menis, and Yelena Beale, Westat

## 1. Introduction

The Blaise Resource Database contains a powerful expression language for identifying items on a page extracting information or customizing appearance or functionality on a page. In this paper we provide more examples of usage, such as improving table presentation to establish and communicate the location of the cell being edited to interviewers and testers. Additional information, such as the data field name, may be provided for the situation where a Data Manager might find it useful, while it could be excluded for an Interviewer who might find it distracting.

For accomplishing specialized tables presentation using the expression editor, State and Page variables are a useful place to start. As the complexity of the instrument grows, so might the need to create expressions to properly filter for the desired item as well as fully identify it. This paper will focus on page level expressions in terms of examples and syntax. The complexity of the datamodel examples will demonstrate the expression language can disentangle the block+Field name for an arrayed cell name for useful results.

The expression language also has many useful features that help in controlling the operation of actions and properties, many under-valued, and we examine examples such as inspecting data for multiple fields on a page.

## 2. Displaying Informational Text

The table below (Figure 1) contains a roster of persons and multiple demographic questions to be asked about each person. We identify the active field and wish to display in a simplified view the subject (i.e., the row ID) of the question, as well as some metadata to indicate what question is being asked.

**Figure 1.** Roster with current row and its active field information shown in yellow background.

The screenshot shows a Blaise 5 interface window titled "CAPI roster with identifier information". The main content area is divided into several sections. At the top, there is a blue header bar with the title and an "ENG" button. Below this, the text "NameGrpRoster" and "LocalName: GrpRoster" is displayed. A yellow highlighted box contains the following metadata: "ROW ID: Brian", "COLUMN HEADER {Role=Description}: Gender", "ActiveFieldName: Roster[2].Gender", "Page Name: Roster[2].Gender", and "Page LocalName: Gender". Below the metadata, a question "What is Brian's gender?" is displayed. Underneath the question is a table with five columns: Identifier, DOB, Age, Gender, and Marital Status. The table contains three rows of data. The second row, corresponding to Brian, is highlighted in yellow. The "Gender" cell in this row is circled in red. At the bottom of the interface, there are two navigation arrows.

Identifier	DOB	Age	Gender	Marital Status
Alice	3/2/1998	35	Female	DIVORCED
Brian	12/12/1985	39	Male	SEPARATED
Charles	4/23/2020	4	Male	NEVER MARRIED

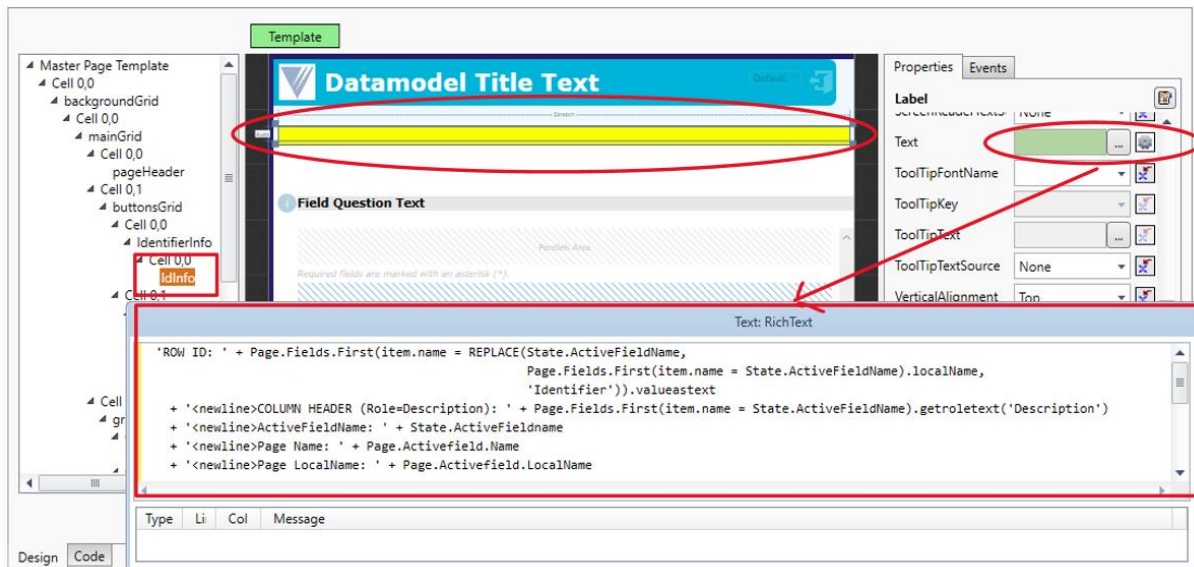
## 2.1 Layout Templates

The MasterPage template contains a label IdInfo. The roster uses a customized Table Template. The subject's information is being displayed in the label. The label's text property utilizes the expression language to build the five lines of display text. The second row of the table is the current active row and the Gender column is the "active field."

## 2.2 The Expression

The IdInfo label on the MasterPage template and its Text property expression are seen in Figure 2 below. It will resolve to display five lines of text.

**Figure 2.** Label control IdInfo has a text expression to be evaluated and shown at runtime.



As noted, the IdInfo text property value is determined by an expression, but it may help to think of this expression result being built from five individual expressions – one expression for each line and where a <newline> tag is used trigger the beginning of each individual new line.

1. The ROW ID shows the name of the person of the current row – that is the person being asked about. By way of deconstruction of the expression, we have Page variables and properties:
  - Page.Fields.First() identifies the first field on a page that meets a specified condition. The .Fields indicates the items we are looking on the page are fields. The .First() method is one of the standard collection of methods and properties and it identifies the first object in the collection – in this case the collection of fields on the page – where the field's name (i.e., item.name) that meets the conditional expression *item.name = REPLACE(S, Old, New)*. Again, **item** is a keyword that may be thought of as a variable referring to an individual object in the collection. The REPLACE function arguments will be discussed shortly, but for the item (Field) that has its name matching whatever is returned in the REPLACE expression, we will consider this field as the "first" field and its properties are the ones of interest. Note: if there is no condition (i.e., there is nothing between the parenthesis for .First(), then the first field on the page is chosen.
  - We are looking for the Identifier column's value of the row that is the current row. We know the Localname of the column is 'Identifier', but we also need the BlockName. State.ActiveFieldName contains the name of the current field and this name includes both the

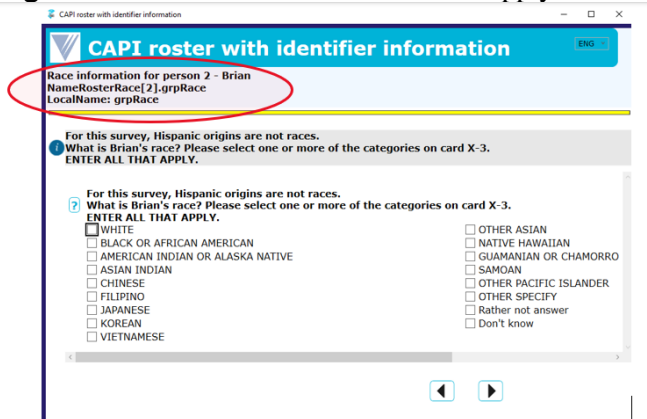
- BlockName and the LocalName components. This ActiveFieldName is **S** in the REPLACE function.
- The LocalName of the first Field is available to us from ActiveFieldName. The Page.Fields.First(item.name = State.ActiveFieldName) identifies this Field and so once we have the Field, we obtain the LocalName property directly with Page.Fields.First(item.name = State.ActiveFieldName).LocalName. This is the Old parameter in the REPLACE function.
  - We are looking for the data associated with the Identifier column – that is the current row’s person’s name. If we replace the LocalName part of the ActiveFieldName with ‘Identifier’, we will have a fully qualified BlockName + ‘Identifier’ – that is current row’s full fieldname.
  - Once we have the full name for the Identifier field, we simply use the ValueAsText property to get this Identifier field’s value and so we have a name of the person for the current row.
2. Line 2 displays the roster table’s column header text for the active field. By way of design, the column header is being stored in the ‘Description’ role text.
    - Like in the line 1 expression, we use the Page.Fields.First() construct to obtain the current field of interest and the GetRoleText method to get the desired text.
    - The (*item.name = State.ActiveFieldName*) -- The condition has the item object’s name must match the active field’s name.
    - .GetRoleText('Description') This part of the expression obtains the role text where the Role is Description. Note: The roster table uses ‘Description’ role. Other tables may use something else, such as the Name of the field.
  3. Line 3 State.ActiveFieldName is the full name (Block-path and localname) of the current field Blaise has on the route.
  4. Line 4 Page.ActiveField.Name – Page has a variable called ActiveField and the .Name method returns the full name
  5. Line 5 Page.Activefield.LocalName – The Page ActiveField also includes the LocalName method to return the field name w/o block information.

### 3. Displaying Group Information

In our next example we have a roster of persons, with each person being asked demographic questions. In this situation, we have a group of questions.

In the page screenshot below (Figure 3), we identify the active field and wish to display in a simplified view, who is the subject (i.e., the row ID) of the question and some metadata to indicate what question is being asked. The goal to be met requires construction of expressions that result in displaying this information in a label on the page.

**Figure 3.** Race is asked as code-all-that-apply with an other-specify.



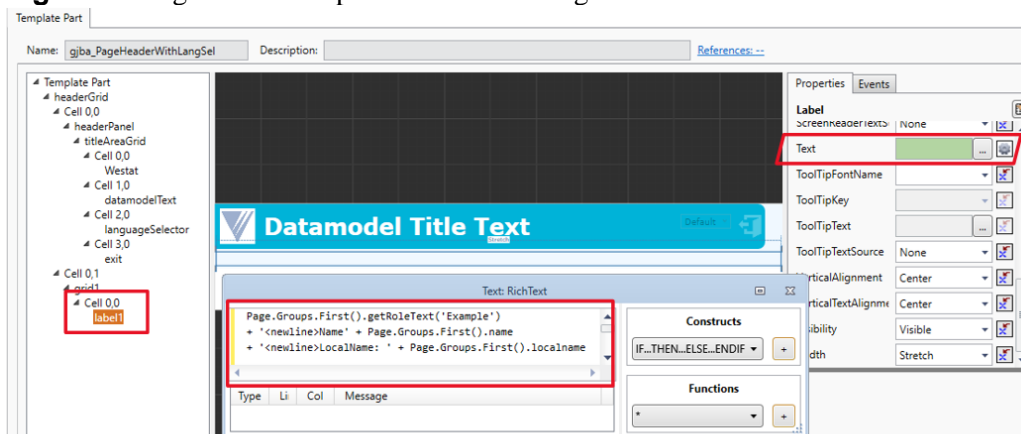
### 3.1 Layout Templates

The Masterpage has a PageHeader template part containing a label. The content of the page uses the OtherSpecify group template. Here we provide header section as part of the PageHeader where role text provides detail about who is the subject, which iteration of the loop of the person roster the survey is on as well as on the second line, the group name (Block and GroupName) and on the third line, just the LocalName of the group.

### 3.2 The Expression

Group information may be used in an expression as part of the expression language. It is important to know that some sort of group template such as OtherSpecify or specialized template such as a section template needs to be applied for Blaise 5 group expression variables to be activated. In this example, we added a text control (label1 in Figure 4.) to a Template part associated with the MasterPage. This label will have its text property using an expression to display what is in Figure 3.

**Figure 4.** Page header template for a MasterPage – note Label1 label control



As one may see, the Text property has three lines.

- Line 1 -- Page.Groups.First().GetRoleText('Example') This line gets the first group on the page and the role text associated with the Role Example. Below is a screenshot of the metadata for the example 2 where the group is grpRace:

**Figure 5.** MetaViewer view of group role text.

General	
Name:	grpRace
Texts	
Description:	(ENG) 'Race group'
EXAMPLE:	(ENG) 'Race information for person ' + pOrder + ' - ' + Identifier
Question:	(ENG) 'Race information'

- Line 2 '<newline>Name' + Page.Groups.First().name *This returns the full block + LocalName of the group*
- Line 3 '<newline>Name' + Page.Groups.First().Localname *This returns LocalName of the group.*

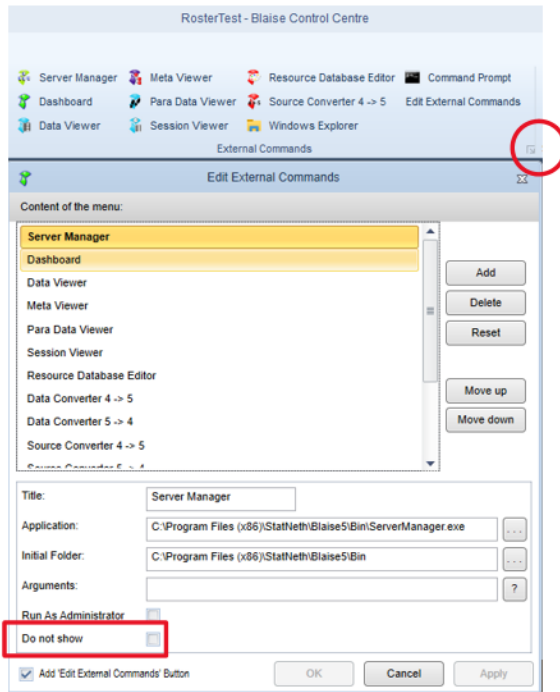
#### 4. Concluding Remarks

Expressions are powerful, but they can be a challenge. The on-line help is a good start, but more examples will always be helpful. Having a sense of the metadata may be helpful, especially in the sense of the type of object (e.g., category vs. field vs. group, etc.) that one is trying to process. Being aware of how the instrument is programmed, with elements like role text and groups, is also important. In the end, one may need to consult with Team Blaise with some constructs – the issue may be the syntax or it may be the way the instrument was programmed.

## Appendix 1 – The MetaViewer tool

Configuration for the Metadata Viewer to appear in the Blaise 5 control center is straightforward - see Figure A1. below. To access the “Edit External Command” form, click on the expansion control (circled in red in Figure A1. Once shown, select the Meta Viewer listed and make sure the “Do not show” is unchecked (red box in Figure A1.)

**Figure A1.** Blaise 5 Control Centre middle section of the ribbon containing external commands



To run the MetaViewer tool, one may now simply select it from the ribbon and use the File Menu to open the desired .bmix.

For example, we show what one may see for the given survey instrument being discussed in the page. In particular, we may see the grpRace group identified. See Figure A2.

**Figure A2.** MetaViewer view of part of the RosterTest datamodel.

