

## SCREENDRIVER Documentaion

Documentation        Last Update: APR 14 2005  
Program:                BQL.SCREENDRIVER  
Programmer:            Chris McIntyre

This document describes the program BQL.SCREENDRIVER and its associated subroutines. It is intended to be the bulk of, and a supplement to, the documentation and comments that would otherwise be included directly in the source code. Source code comments exist and are encouraged, but the bulky comments are put here to reduce program clutter. File layouts for all of the files involved can be found in separate documentation.

### OVERVIEW

BQL is a screen generator. The program BQL.SCREENDRIVER uses pre-tabulated data to present and drive a data-entry screen, referred to here as the "BQL Screen". The BQL screen is designed to be similar to familiar data entry screens currently used by REI in the PICK environment. The pre-tabulated data tables are "constructed" by a programmer or other individual before run-time, amended as needed, and used on demand when the BQL screen is run.

A "database" is one file whose records are edited by the BQL screen. Information particular to a database is stored in the file BQL.SYSTEM. Item ids in BQL.SYSTEM are simple sequentially generated numbers that contain no data. Orininally the BQL screen was designed to edit only one item at a time within one database at a time without juggling multiple items or multiple files. Currently multiple items may be presented on the screen but requires supplemental programming. The variable "DBNUM" is used in BQL.SCREENDRIVER to store the item id of the database.

The BQL.SCREENDRIVER editing session may employ several screens to accomodate database record layouts that have lots of fields. Currently there is an arbitrarily set limit of 64 screens in a record editing session. There is no limit to the number of fields in a record that can be set up for editing other than the practical limit of what can be squeezed into the 64 screens.

The variable "SCR" is used to store the current screen number. The variable MAXSCR is used to store the highest screen number. MAXSCR is set automatically.

Field numbers must be entered directly into the field definition items before run time. If fields are added, deleted, or moved the fields need to be renumbered. "By hand" this would be burdensome but in

practice a utility can be used to make this very easy. At this time, such a utility is connected to BQL.SCREENDRIVER (on DEVEL but not LIVE) to automatically renumber (whether needed or not) field items each time the screen is invoked.

Field numbering on the screens can be done in different ways. One option is to set all screens in a session to have their numbering begin with 1. Another option is to begin field numbering on one screen as a continuation in sequence from the previous screen. Alternatively, overrides can be established for field numbering allowing any kind of numbering scheme the constructor cares to set up. The field numbers appearing on the screen don't need to be consecutive. Gaps in numbering are allowed with the system automatically keeping track of the cursor sequencing.

The variable "FLU" is used to store the value of the field number that the user sees on the screen. In multiple-screen sessions, an external field number (FLU) may be repeated across many screens, thus the variable FLU does not necessarily specify a unique field in the record. The variables SCR and FLU together do however specify a unique field. Internally, lots of things particular to the record's fields are stored in dynamic array variables and are indexed using the variable "FLD".

BQL.SCREENDRIVER is designed especially to allow quick and easy additions and deletions of fields, quick re-positioning of fields on screens, and quick transfer of fields to different screens.

The BQL Screendriver can't everything so it does not try to. Many situations and scenarios will arise in the data entry screen environment that are beyond the scope of the BQL Screendriver. Simple parameterization of run time data can't anticipate and provide logic for every situation. For this reason, hooks to external subroutine calls are provided at various points in the screen's logic sequence using PICK BASIC's CALL @variable feature. Currently there are 37 such "CALLPOINTS". Examples of places where callpoints are provided in code are...

- BQL.SCREENDRIVER start up
- BQL.SCREENDRIVER shutdown
- Beginning of each field entry
- Ending of each field entry
- Field data entry validation-time

Several files are used in the Screendriver. File Specifications for all files are stored in a separate document. You're encouraged use the file specs together with this overview to familiarize yourself with the system. Here's an overview of some of the more important and more frequently used files.

BQL.SYSTEM is the "centerpiece" file. An item in BQL.SYSTEM contains data for one database (one specific file). The one most important data item in this file is the file name.

BQL.FLDINFO is probably the "busiest" file. An item in BQL.FLDINFO contains data specific to one field. Generally there is one BQL.FLDINFO item for each single-valued field in the data entry screen(s). Examples

of data in this field are X,Y cursor positions, field widths, the text of field prompts, data types, etc.

Two other files, BQL.MULTI1 and BQL.MULTI2 are used to contain the data specific to a set of multi-valued fields that are displayed and edited on the screen(s).

## DATA TYPES

A concept central to BQL.SCREENDRIVER is that of "data types". A data type indicates what kind of data is being captured in a field, ex. text, integer, date, file-lookup, etc. Without exception, every field on the screen must have a data type assigned to it. Data types dictate how information typed into the screen by the user is to be validated, and how it is to be converted for internal storage. The routine BQL.VALIDATE is used to indicate if data entered in a field is acceptable input, and it relies on the field's data type to decide what validation rules to use.

For convenience the data types are actually organized into broad "DATA CATEGORIES" and then sub-divided into "DATA TYPES". Data Category 1 (standard data) defines many of the common types and when used saves lots of programming. If you can not find a data category/type to suit your need then callpoints are provided to let you write your own validation routine. Some examples of data categories and types...

Data category # 1: STANDARD DATA

Data types: Text  
Integer  
Date  
Flag  
Money  
SSN  
email address

Data category # 2: FILE LOOK-UPS

Data types: GROUPS file lookup  
MEMBERS file lookup  
ENC.HIST file lookup

Data category # 3: SIMPLE TABLES

Data types: PARAM, item = MAC.SAM.RSNS

Data category # 4: USER DEFINED

Data types: various, constructor must write supplemental code.

## ANCHOR POINTS AND FIELD POSITIONING

Cursor and field positioning are done using "anchor points". Refer to the following partial screen dump showing three fields as an example.

1            2            3            4            5

012345678901234567890123456789012345678901234567890

```

0 SCR.nn/nn SCREEN.TITLE.XXXXXXXXXXXXXXXXXXXXXX
1 -----
2
3 1. NAME      XXXXXXXXXXXXXXXXXXXXXXXXXXXX
4 2.  DOB      11-25-1960
5 3.  GENDER   XXXXXXXX
6
7

```

The screen display of fields at run time is organized by breaking them down into three components referred to as "number", "prompt", and "data". In the example above, the Date of Birth field has the character "2" as its number component, the characters "DOB" as its prompt component, and the characters "11-25-1960" as its field component. Each component has its own display parameters to specify the (x,y) cursor position, field width, field-fill character, etc. This makes each of the three components independantly controllable for position and display flexibility.

Even though all three have their own (x,y) positions, none are stored as an "absolute" position giving an exact location. Instead they are stored as a relative offset from a fourth, common (x,y) location referred to as the "ANCHOR POINT". Continuing the "2 DOB" example above by adding an anchor and offsets, the anchor point would be (0,4), the "number" component offset would be (0,0), the "prompt" component offset would be (4,0), and the "data" component offset would be (12,0). Offsets can be negative.

This scheme allows flexibility during screen construction while moving around all the components of all the fields on a screen. Screendriver program logic then simply reads the anchors and offsets, and converts them to an absolute position format for use at run time.

Here's a tabulation of how the field parameters might look for the three example fields above; NAME, DATE, and GENDER.

	FLD#	ANCHOR_X,Y	NUMBER_X,Y,JUST	PROMPT_X,Y,JUST
DATA_X,Y,JUST				
NAME	1	0,3	0,0 R#2	4,0 L#4 12,0
L#25				
DOB	2	0,4	0,0 R#2	4,0 L#3 12,0
R#10				
GENDER	3	0,5	0,0 R#2	4,0 L#6 12,0
L#6				

SECONDARY DISPLAY  
Need some documentation here.

Parameter List  
<1> DBNUM Database id number

<2> UPD.PASS            Update flag  
<3> BUFFID.PASS        Pre-determined Item id  
<4> BUFFID.PASS.NEWOK   Flag controlling if new items are ok  
<5> HUSH.ERR            Flag to suppress system error display

FILES:

BQL.SYSTEM  
BQL.SCRINFO  
BQL.FLDINFO  
BQL.MULTI1  
BQL.MULTI2  
BQL.DATACAT2  
BQL.TABLES

DATA STRUCTURES AND COMMON VARIABLES:

FLD  
FLU  
SCR

FLDINFO.IDS<FLD>      = BQL.FLDINFO.ID

FLDTABLE<FLD, 1>      = SCR  
FLDTABLE<FLD, 2>      = FLU. if null, fld is not numbered  
FLDTABLE<FLD, 3>      = Next FLD for cmode 1  
FLDTABLE<FLD, 4>      = Next FLD for cmode 2  
FLDTABLE<FLD, 5>      = Display only flag  
FLDTABLE<FLD, 6>      = Limited access flag  
FLDTABLE<FLD, 7>      = Blank fill char  
FLDTABLE<FLD, 8>      = Field entry mode  
                      1 = "normal"  
                      2 = scrolling line  
                      3 = mini word proc  
FLDTABLE<FLD, 9>      = Previous FLD to jump to w/ '/' logic  
FLDTABLE<FLD,10>      = conversion to automatically apply to input  
FLDTABLE<FLD,11>      = lookup sample limit override

FLDTABLE1<FLU> = multi value list of SCRs that this FLU appears on  
FLDTABLE2<FLD> = multi value list of FLDs having the same  
BUFF,ATT,VAL,SVL  
                      IOW, duplicated fields.  
                      Aat this time (09/03), for field type 'S' only)

FLDINDX1<SCR,FLU> = FLD

SCRTABLE<SCR, 1>      = FLU number minimum  
SCRTABLE<SCR, 2>      = FLU number maximum  
SCRTABLE<SCR, 3>      = FLD number minimum  
SCRTABLE<SCR, 4>      = FLD number maximum  
SCRTABLE<SCR, 5>      = securtiy access flag

SCRTABLE<SCR, 6> = BQL.SCRINFO.ID  
SCRTABLE<SCR, 7> = Screen header to override system header  
Note: pgms assume that the FLD numbers for a SCR are contiguous ! Be careful.

HELP1.MSGS<FLD> = HELP1 message (fld type = 'S' only)

\*\*\*  
\*\*\*  
\*\*\*

#### BUFFERS AND DATA STRUCTURES USED FOR THE SCREEN'S DATA DISPLAY:

DISBUFF1 is a dynamic array. Widely used, very important. It is used to store a ready copy of the already externally formatted value of all fields. This is handy when for example you are scrolling from subscreen to subscreen and are not updating values but need to constantly repaint entire screens, the external value is already there in DISBUFF1 and does not need to be continually OCONVERTed.

DISBUFF2 is a dimensioned array. It holds secondary display values. Secondary display values are display-only fields that are a function of an entry field. An example would a display-only field showing a MEMBER's name when the MEMBER's number is captured in an entry field.

DC2.TABLE1 is a table of data that organizes information about all the data category 2 files that are used in the screen. Only one entry is made in DC2.TABLE1 for each data category 2 type/file used in the screen. This is to eliminate redundant entries when the same data category 2 type/file is needed for several fields. Attribute 1 stores data category 2 types, attribute 2 stores the type's file name.

DC2.FILES is a dimensioned array. Indexed similarly to DC2.TABLE1, it holds the opened file pointer variables of the files listed in DC2.TABLE1.

DC2.BLDITEMS is a dimensioned array holding items from the file BQL.DATACAT2 that are used in the screen.

DC2.TABLE2 is a dynamic array. It lists the index# into DISBUFF2, DC2.TABLE1, DC2.BLDITEMS, and DC2.FILES for every field on the screen.

DC2.RUNITEMS, a dimensioned array, is obsolete and no longer used. (Replaced by DISBUFF2). There's still (as of 10/04) a reference to this in BQL.COMMON but the array is not used anywhere. Next time you change BQL.COMMON you can delete it.

DISBUFF1<FLD> = The FLD's external display value.  
DISBUFF2<FLD> = The FLD's secondary display values if used/defined.  
DC2.TABLE1<1,X> = list of data category 2 types.  
DC2.TABLE1<2,X> = list of data category 2 file names.

DC2.FILES(X) = opened file pointer variable for the file in  
DC2.TABLE1<2,X>  
DC2.BLDITEMS(X) = the item read from BQL.DATA CAT2 using id in  
DC2.TABLE1<1,X>  
DC2.TABLE2<FLD,1> = index pointer into DC2.TABLE1, DC2.BLDITEMS, and  
DC2.FILES  
DC2.TABLE2<FLD,2> = index pointer into DISBUFF2  
DC2.TABLE2<FLD,3> = Attributes 3 and above previously used but now unused  
and available.

MENUBAR.XYPOS = @(x,y) of MENUBAR starting point  
MENUBAR.CLEAR = clear to end of line from xy pos  
MENUBARS<n> = dynamic array of menus at bottom of screen  
MENUBAR = variable that indexes MENUBARS  
MENUBAR.MAX = maximum MENUBAR value

A group of fields on the screen formed from one or more multivalued attributes in the record being edited is referred to as a "Multi-Valued Window". Variables and structures associated with these windows use the text string "MVW" in their names. Depending on the construction of the screens, there may be any number (including 0) of MVW windows. There is no limit to the number of MVW windows in the database. There may be any number (including 0) of MVW windows on any one of the database's screens.

MVW Identifies the current MultiValue Window

MVW.DATA lists general info on MVWindows, listed by MVW.

MVW.DATA<MVW, 1> BQL.MULTI1 item id  
2> Width (maximum MVX)  
3> Depth (maximum MVY)  
4> MVW.BASEX, anchor point x coordinate  
5> MVW.BASEY, anchor point y coordinate  
6> Minimum FLU number  
7> Maximum FLU number  
8> Controlling attribute  
9> DCOUNT of controlling attribute  
10> SCR  
11> Print attributes flag  
12> Insertions OK flag  
13> Deletions OK flag  
14> Pre-set attributes high flag  
15> additional attributes list (msv)

MVW.INDEX1 lists all MVWs by screen. (static)

MVW.INDEX1<SCR> = MVW y MVW y MVW ...

MVW.INDEX2 lists various MVW data by FLD. (static)

MVW.INDEX2<FLD, 1> = MVW  
2> = MVX  
3> = MVY  
4> = more??

MVW.INDEX3 indexes the fields by MVW, MVY, and MVX. (static)  
MVW.INDEX3<MVW, MVY, MVX> = FLD

MVW.INDEX4 lists column specific data. (static)  
MVW.INDEX4<MVW, MVX, 1> = attribute  
2> = offset  
3> = HELP2 messages  
4> = FOOTER message  
5> = RBUFF(n) buffer number

MVW.INDEX5 keeps track of dynamic row data. (scrolling values)  
MVW.INDEX5<MVW, MVY> = value

MVW.INDEX6 contains data specific to the column of displayed  
scrolling value-mark values  
MVW.INDEX6<MVW, MVY, 1> = XY01 ( @(xy) pos of displayed value)  
2> = JFW (just, fillchar, width)  
3> = width  
4> = wrap characters  
5> = X display coord  
6> = Y display coord

MVW.INDEX7<SCR> = current default MVW

#### CALLPOINTS...

02	FLDINFO	<54,05>	Post field processing
03	SYSTEM	<27>	Alt external subr to auto-generate an item id, retired 4/05
04	FLDINFO	<54,01>	Supplemental Field entry validation
05			Supplemental lookups (see db 121)
06	SYSTEM	<29,07>	Post initialization
08	SYSTEM	<29,05>	After generating FS, before PRINTing the FS
09	SYSTEM	<29,01>	Pre filing record validation
10	FLDINFO	<54,02>	Supplemental check to see if field access allowed
11	SYSTEM	<29,02>	Pre filing
12	SYSTEM	<29,03>	Post filing
13	SYSTEM	<29,04>	After rec has been read, before editing begins
14	FLDINFO	<54,06>	After KEY1 entry
15	SYSTEM	<29,06>	Initialization begin
16	SYSTEM	<28>	Alt external subr to prompt for, lookup, valid item id
17	SYSTEM	<29,08>	Post record read, Post BLOAD, pre-edit (need this on screen!)
18	SYSTEM	<29,09>	After CALL to BQL.GETRUNDATA
19	SYSTEM	<29,10>	Pre main prompt
20	SYSTEM	<29,11>	Post getid, pre record read (see db 59)
21	DATA CAT4	< 2 >	Data category 4 validation routine (user defined)
22	DATA CAT4	< 3 >	Data category 4 iconv routine, (unused, iconverting is done within the validation process)

23 DATACAT4 < 4 > Data category 4 oconv routine (user defined)  
 24 SYSTEM <31,nn> Z Commands  
 25 SYSTEM <29,12> Default Set-up  
 26 SYSTEM <29,13> Modify KEY1  
 27 FLDINFO <54,07> Post validate, pre-display  
 28 SYSTEM <29,14> Alternate record exit prompt (when 'X'ing out)  
 29 SYSTEM <29,15> Modify MIP (the Main InPut variable)  
 30 SYSTEM <29,16> Supplemental item ID validation  
 31 SYSTEM <29,17> After Background Screen Generation  
 32 SYSTEM <29,18> Before Print Screen  
 33 SYSTEM <29,19> After Print Screen  
 34 SYSTEM <29,20> Alternate routine to get ptr# in Print screen.  
 35 SYSTEM <29,21> Logic to supplement setup of NEED.TO.FILE (when  
 'F'iling out)  
 36 FLDINFO <54, 8> Routine to handle EMODE 4  
 37 FLDINFO <54, 9> Supplemental fld validation w/override  
 38 SYSTEM <55,nn> Get id lookup display  
 39 SYSTEM <29,22> Logic to supplement setup of UPD at initialization  
 time

#### Loose Checklist of Items Needing Pushing at Install Time

your BQL.EXSUBR.nnn program  
 any other programs  
 items in BQL.SYSTEM  
 items in BQL.SCRINFO  
 items in BQL.FLDINFO  
 items in BQL.MULTI1  
 items in BQL.MULTI2  
 items in BQL.SECTABLES  
 items in BQL.DISPLAY.FORMATS  
 items in BQL.DATACATx files  
 constant items  
 q pointers  
 number generator items

(the following is not necessarily a complete list)

Find BQL.SECTABLES	items in BQL.SYSTEM	atts	4	33	36	40	41	42
43								
Find BQL.SECTABLES	items in BQL.SCRINFO	atts	11					
Find BQL.SECTABLES	items in BQL.FLDINFO	atts	41					
Find BQL.SECTABLES	items in BQL.MULTI1	atts	41					
Find BQL.SECTABLES	items in BQL.MULTI2	atts	41					
Find BQL.DISPLAY.FORMATS	items in BQL.SYSTEM	atts	70					
Find BQL.DISPLAY.FORMATS	items in BQL.SCRINFO	atts	?					
Find BQL.DISPLAY.FORMATS	items in BQL.FLDINFO	atts	?					
BQL.DISPLAY.FORMATS	items in BQL.MULTI1							
Find BQL.DISPLAY.FORMATS	itmes in BQL.DATACAT2	atts	8	11				
Find BQL.DISPLAY.FORMATS	itmes in BQL.DATACAT3	atts	3					
Find BQL.DISPLAY.FORMATS	itmes in BQL.DATACAT5	atts	5					
Find BQL.DISPLAY.FORMATS	itmes in BQL.DATACAT7	atts	6					

examples...

```
                1          2          3          4          5          6          7
0123456789012345678901234567890123456789012345678901234567890123456789012
3456789
```

```
0 SCR.nn/nn  SCREEN.TITLE.XXXXXXXXXXXXXXXXXXXXXX  (U)Recid:
RECORD.IDXXXXXXXXXXXXXXXXXXXX  0
1 -----
-----  1
2
0
3
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0 -----
-----
1 Select: xxxxx File  eXit  ?Help  More
2 -----12-20-01 cmcintyr
Port_118 OCI.HC
3 PRINTERR line
```

```
0123456789012345678901234567890123456789012345678901234567890123456789012
3456789
                1          2          3          4          5          6          7
```

Here's an example of a Multi Value Window (MVW)...

```
0 SCR.nn/nn  SCREEN.TITLE.XXXXXXXXXXXXXXXXXXXXXX  (U)Recid:
RECORD.IDXXXXXXXXXXXXXXXXXXXX  0
1 -----
-----  1
2     MEMBER_ID          nnnnnnnnnnnnnnnnnn  LASTNAME,FIRSTNAME_____  DD-
DD-DDDD          0
3  1. DOG'S NAME          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3
4  2. DOG'S DOB:          dd-dd-dddd
5  3. INSURED? Y/N:          xxx
6  4. NUMBER OF KIDS:          nnnnn
7
8
9
0
1
2     BEGIN.DATE      ENDING.DAT      RIDER.PACKAGES
3
4  5. 01  dd-dd-dddd  dd-dd-dddd  xxxxxxxxxxxxxxxxxxxxxxxx
5  8. 02  dd-dd-dddd  dd-dd-dddd  xxxxxxxxxxxxxxxxxxxxxxxx
6 11. 03  dd-dd-dddd  dd-dd-dddd  xxxxxxxxxxxxxxxxxxxxxxxx
7 14. 04  dd-dd-dddd  dd-dd-dddd  xxxxxxxxxxxxxxxxxxxxxxxx
8
9
0 -----
-----
1 Select:          File  eXit  ?Help  More
2 -----12-20-01  cmcintyr
Port_118 OCI.HC
3 PRINTERR line
```

... MVW example continued:

The twelve fields numbered 5, 8, 11, and 14 above give an example of a multi-value window. (Fields 1 thru 4 are for illustration and are irrelevant to this example). The example has three columns and four rows. (Refer to file layouts of BQL.MULTI1 and BQL.MULTI2.) Data specific to the entire window are stored in BQL.MULTI1. Data specific to a particular column are stored in BQL.MULTI2. One specific x,y cursor coordinate on the screen is designated at the window's "anchor" point. All other x,y coordinates for things in this window are given as an offset from this point. When the screen layout changes or the fields are rearranged, a simple change to the anchor point is all that is needed to move the entire window. The anchor point might typically be in the upper left corner of the field numbers. In this example that would make the x,y anchor point be 0,14.

Columns display data in one attribute. Rows display data in one multi-value. The window scrolls up and down. In order to view multi-values that don't fit in the window, the user can issue scroll commands to move data into view. The window does not scroll left and right to display other attributes.

Field numbering comes in two flavors. 1) Field numbers are displayed only for fields in the leftmost column. All other editable fields in the window have sequential field numbers, but their display is suppressed and the user must interpret their value. This column of field numbers is established and incorporated into the BS variable. 2) Field numbers are displayed only for fields in the leftmost column, but the other editable fields have no displayed or implied numbers.

The column of 0-padded numbers just to the right of the field numbers is a display of the multi-values currently in display in the window. Display of this column of numbers is dynamic - it will change if and when data in the window is scrolled. This display is stored in the variable MVW.BS. Displaying these multi values can be suppressed.

The variables MVX and MVY are used in the screen driver program to identify the columns and rows respectively in the window. In this example, the lowest, rightmost field would be in column 3 (MVX = 3) and in row 4 (MVY = 4).