# RELATIONSHIP AMONG PAGES, VIEWS, FIELDS AND SCREENS

**Thomas E. Berendt, * Internal Revenue Service**

## Abstract

SQL*Forms 3.0 provides great versatility for creating a window- like environment via pop-up pages. However, a developer who is unfamiliar with all of the available features and procedures may spend some time using the trial-and-error method in order to get pop-up pages to pop up when and where they are needed. In fact, this was the method which I used while developing my first few forms.

This paper shows how pages and views of pages can be used in the development of screens in SQL*Forms 3.0. The size and placement of pages and views are initially set in the Page Definition Table but can be changed using the packaged procedures, ANCHOR_VIEW, MOVE_VIEW, RESIZE_VIEW, HIDE_PAGE and SHOW_PAGE.

A method for setting up fields to look like page text (via Oracle*Terminal) is also shown. This eliminates the need to create different pages for the sole purpose of varying text portions of a page.

## Introduction

To a large extent, my early difficulties with pages, views and screens resulted from not having a clear understanding of what they were. For this reason, I will begin with descriptions for these items. Next, I'll go over the characteristics of the Page Definition Table. Then I'll demonstrate how some of the packaged procedures work, along with some examples from my own applications. Finally, I will show, step-by-step, how to use Oracle*Terminal to set up a video attribute which can be used by SQL*Forms to change a field to look like text.

## Descriptions

### Page

A page is basically a rectangular object which contains fields and text. Fields from more than one block can be on the same page. If the cursor is positioned at one of the fields of a page, then that page is considered to be the active, or current, page.

### View

A view is a subset of a page. It can include the whole page or any rectangular part of a page. The view is what will appear on the screen when its page either becomes active or is the argument of the SHOW_PAGE packaged procedure. In the former case, if the field at which the cursor is positioned (aka SYSTEM.CURSOR_FIELD) is not within the view of the page, then the view will automatically be moved so that this field will be in the view. In the latter case, the view will appear "beneath" or "behind" the view of the active page (i.e., the view which is displayed by the SHOW_PAGE packaged procedure may be partially or completely covered by the view of the active page). No more than one view of a page can be shown on the screen at a time.

### Screen

The screen is the output device. It's where the views are displayed. Several views can be displayed on a screen simultaneously and more than one view will be visible if they occupy different places on the screen. When views are displayed, they are placed over any other views (except for the active view) which occupy the same space on the screen. If a view which lies over another is removed, then the underlying view will reappear.

## The Page Definition Table

Page characteristics are initially set in the Page Definition Table. Figure 1 illustrates one format of the Page Definition Table.

The page number is the first characteristic to be designated. Beware that it cannot be changed once the cursor leaves the row for that page in the Page Definition Table. The only alternative is to remove the page and insert a new one. The Pop Up check box is used to set or unset the pop up characteristic for the page. The pop up characteristic must be set in order to use any of the other page characteristics. Page Size sets the width (X) and the height (Y) of the page. A page

*This paper was originally presented at the 1992 International Oracle User Week Conference in San Francisco, California, September 14-18, 1992.

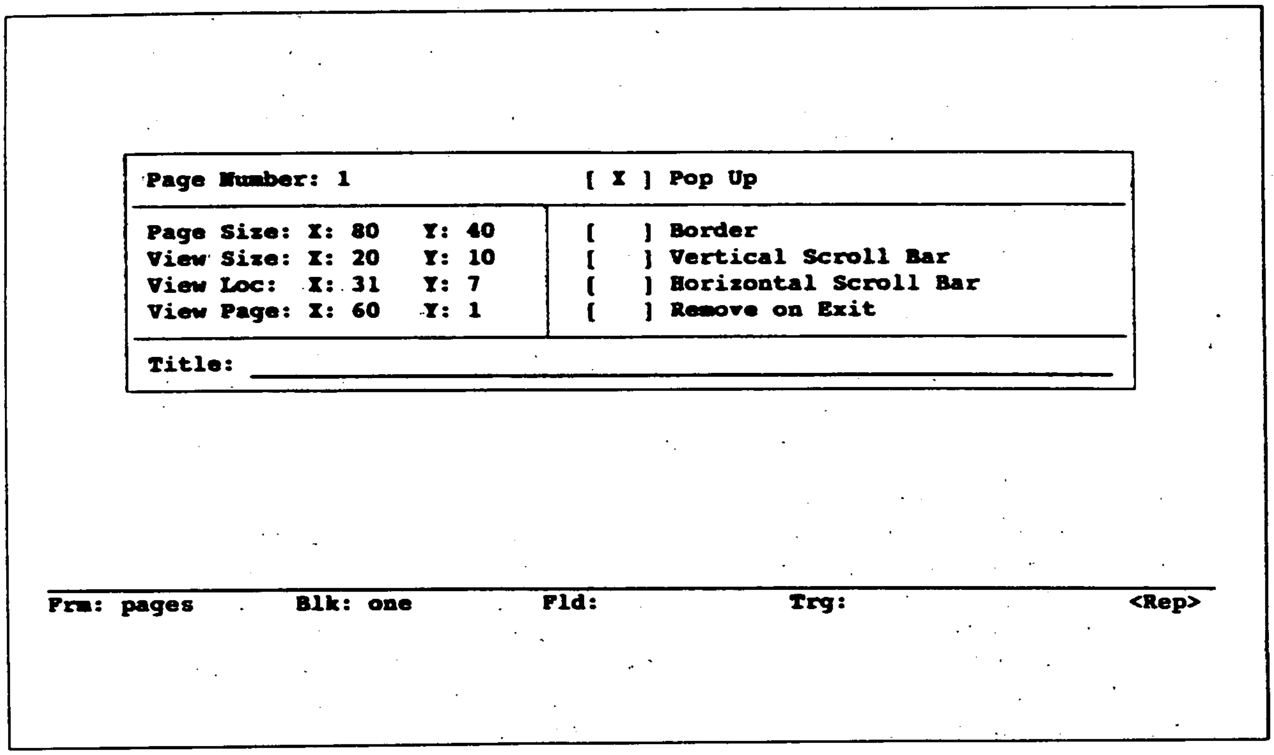
can be a minimum of 1 column wide and 1 row high and a maximum of 255 columns wide and 255 rows high. Here, the page size is 80 columns wide by 40 rows high.

View Size sets the width (X) and height (Y) of the view of the page. Figure 1 shows a view size of 20 columns by 10 rows. The view settings can be changed by the RESIZE_VIEW packaged procedure. View Loc is the initial position of the view on the screen. The upper-left corner of the view will be shown on the screen at the coordinates entered here. Figure 1 indicates that the upper-left corner of the view will be in column 31 of row 7 of the screen. The ANCHOR_VIEW packaged procedure is used to change these coordinates. View Page indicates what part of the page will be in the view. The coordinates entered here will pinpoint the column and row of the page where the view will start (i.e., the upper-left corner of the view). The View Page characteristic of figure 1 shows that the upper-left corner of the view will show column 60 of row 1 of the page.

The MOVE_VIEW packaged procedure is used to change these coordinates. In other words, the View Page characteristic and the MOVE_VIEW packaged procedure select the part of the page to show on the screen; the View Loc characteristic and the ANCHOR_VIEW packaged procedure select where to show this view on the screen. Putting it all together, the Page Definition Table in figure 1 creates a page where the view will show columns 60 through 79 and rows 1 through 10 of the page (unless modified by the use of borders, scroll bars or a title). This view will be shown on columns 31 through 50 and rows 7 through 16 of the screen. Figure 2 shows the screen when the form is executed. The view consists of four lines of text and one 10-character field. The border was put in via the Draw Box feature of the screen painter, not via the Border characteristic. The Draw Box border was included to demonstrate differences between the two types of borders.

The Border characteristic specifies whether or not the view of the page will automatically have a border drawn on its edges. This border will take

Figure 1

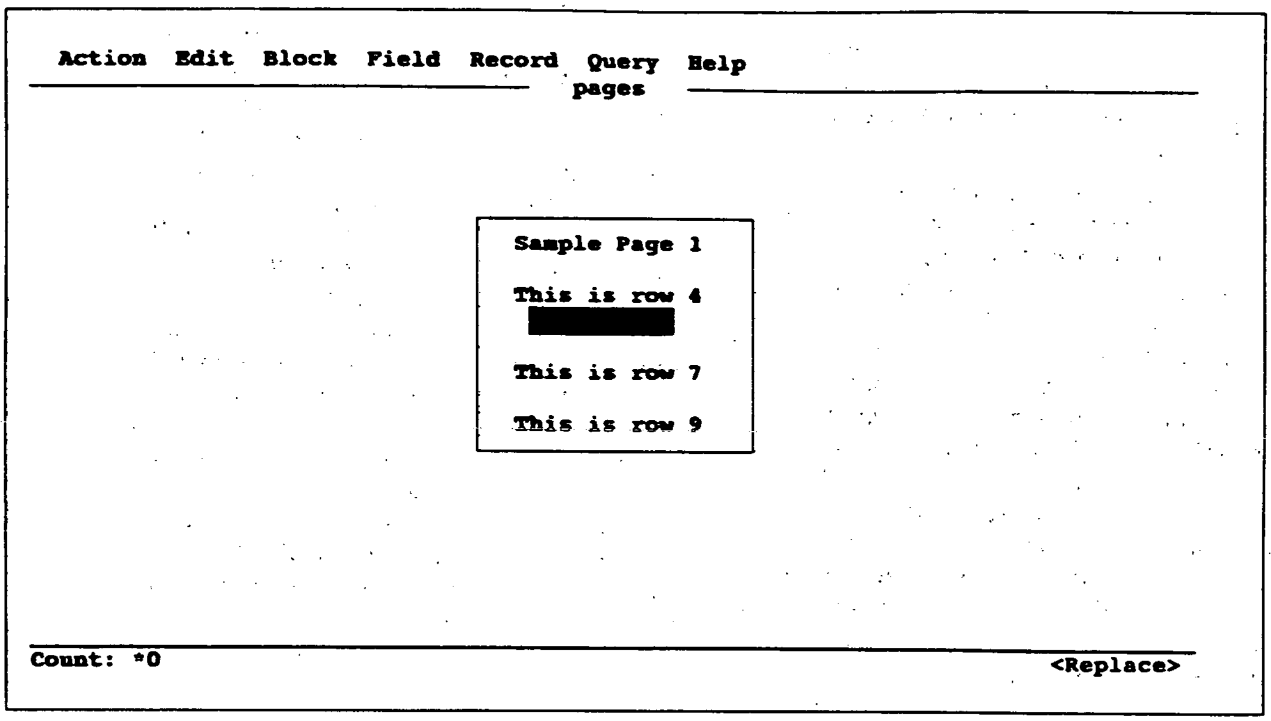

```
Page Number: 1                    [ X ] Pop Up

Page Size: X: 80    Y: 40         [   ] Border
View Size: X: 20    Y: 10         [   ] Vertical Scroll Bar
View Loc:  X: 31    Y: 7          [   ] Horizontal Scroll Bar
View Page: X: 60    Y: 1          [   ] Remove on Exit

Title: ________________________________________

Frm: pages      Blk: one      Fld:      Trg:                <Rep>
```

Figure 2



```
  Action  Edit  Block  Field  Record  Query  Help
  ___________________________________ pages ___________________________

                          Sample Page 1

                          This is row 4


                          This is row 7

                          This is row 9

  _____________________________________________________________________
  Count: *0                                                  <Replace>
```

up part of the view, so less of the page will be visible in the view if a border is selected. Also, the View Page or MOVE_VIEW coordinates will be changed from the upper-left corner of the view to just inside the border. This occurs so that the left and top edges of the view of the page are not overlaid by the border; instead, the last two columns and the last two rows of a borderless view are lost when a border is added. Figure 3 shows the screen after selecting the Border characteristic for the page. Note that the Draw Box border was moved inside of the new border and that part of the bottom and right edges of the view in figure 2 are no longer displayed.

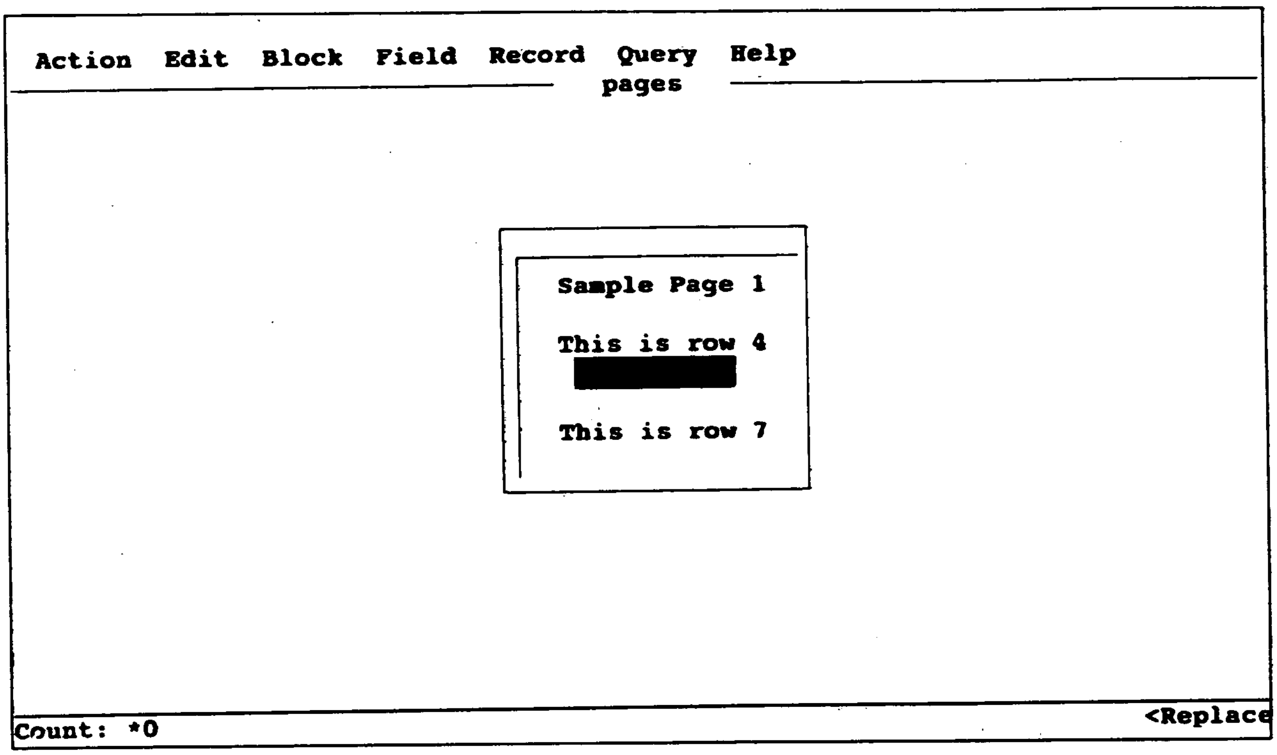
Scroll bars are used to give information to the operator about the relation between the view and the page. If an arrow is present in a scroll bar region, it indicates that the page extends beyond the view in the direction to which the arrow points. The bar itself indicates the relative position and size of the view on the page. The scroll bars will also reduce the amount of the page that will appear in the view. Figure 4 shows the screen after turning on both the vertical and horizontal scroll bars. The bar in the center area of the horizontal scroll bar is near the right edge of this area; this indicates that the view is near the right edge of the page. The bar also takes up 3 of the 12 spaces available in this area; this indicates that the view is showing about 1/4th of the width of the page.

The Remove on Exit characteristic controls the longevity of a view. If this characteristic is set, then, when the cursor leaves the page, the view will disappear. In this case, the view will not exist beneath or behind other views. Remove on Exit has no effect on a page which is displayed via the SHOW_PAGE packaged procedure unless and until that page becomes the active page.

The Title characteristic puts a title on the view. If a title is not desirable, then the space should be left blank. A title will further reduce the amount of the page that will appear in the view. Figure 5 shows the screen after typing "View of Page 1" into the Title characteristic for the page. Note that the upper- left corner of the view no longer appears. The position of the view on the page went through an automatic MOVE_VIEW in order to keep the current field visible. The vertical scroll bar has been shrunk so much that it is almost useless. It does show, however, that the page extends both above and below the view.

Chapter 10 of the "SQL*Forms Designer's Reference Version 3.0" mentions restrictions on the definition of some of the characteristics. If these restrictions are violated, the form may still be generated and executed without any error messages. However, the views may not display on the screen as expected.

Figure 3



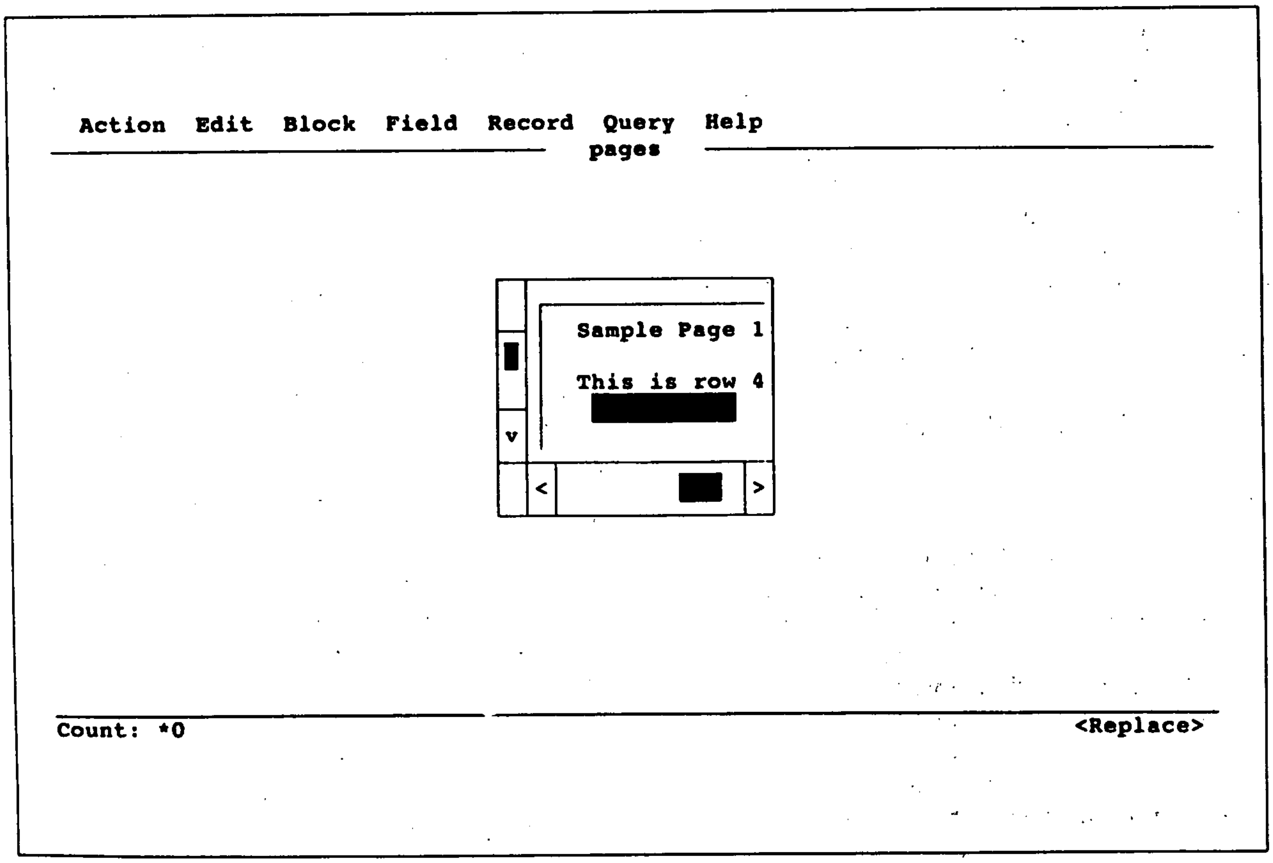
Sample Page 1

This is row 4

This is row 7

Count: *0 <Replace

Figure 4

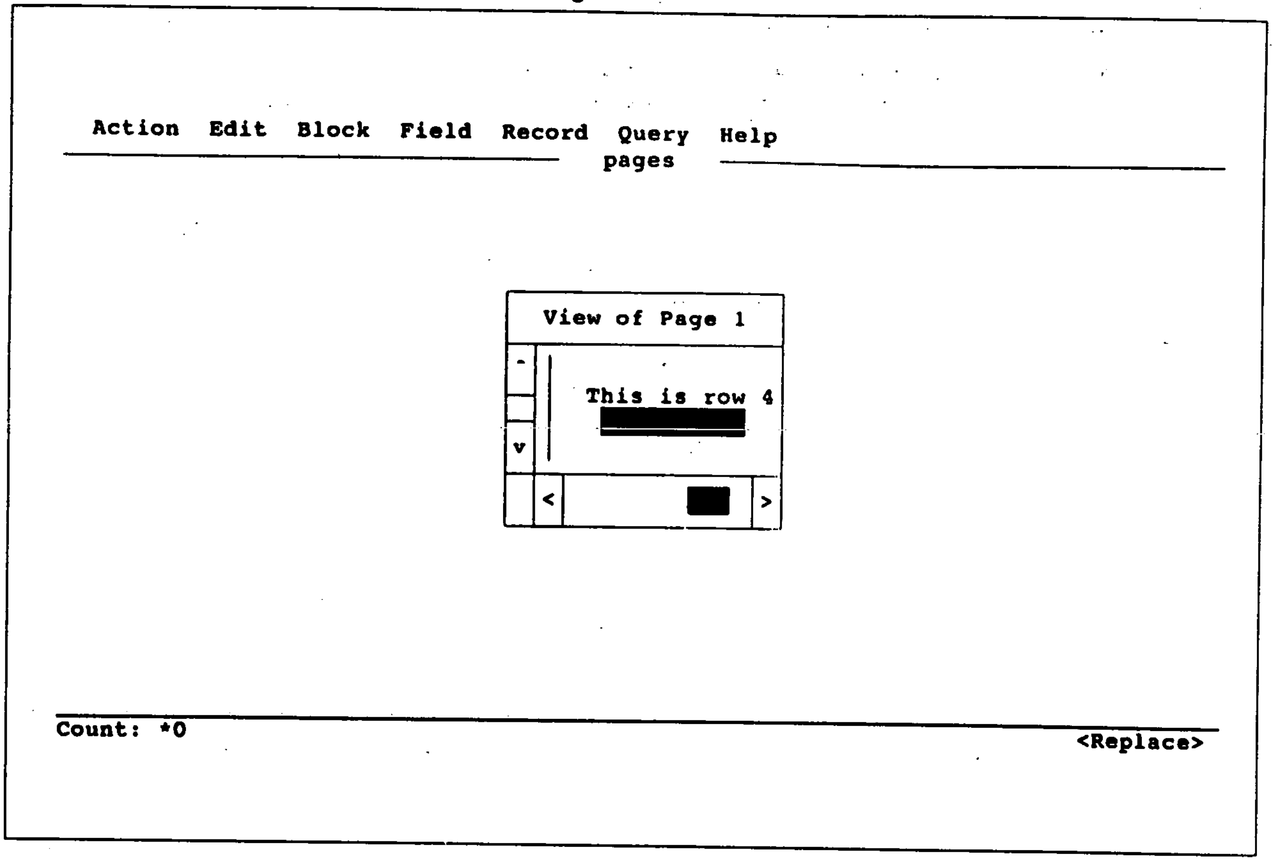


Sample Page 1

This is row 4

Count: *0 <Replace>

Figure 5



## Packaged Procedures

Packaged procedures can be used to make the pages of a form more dynamic. ANCHOR_VIEW moves a view from one place on the screen to another. MOVE_VIEW moves a view from one section of a page to another (i.e., a different part of the page will show on the screen). RESIZE_VIEW changes the size of a view. SHOW_PAGE shows a view on the screen when its page is not current. HIDE_PAGE removes a non-current view from the screen.

Let's add a KEY-HELP trigger to our form with the code:

```
ANCHOR_VIEW(1,10,8);
```

When the trigger is fired, the screen will look like figure 6. The view's "anchor" is its upper-left corner. The ANCHOR_VIEW causes the view to "lift anchor" from its current screen location, "sail" to the new location and "drop anchor." In this case, the view has been "re-anchored" at column 10 and row 8 of the screen. Note that only the location of the view on the screen has been changed. The contents of the view is the same as before.

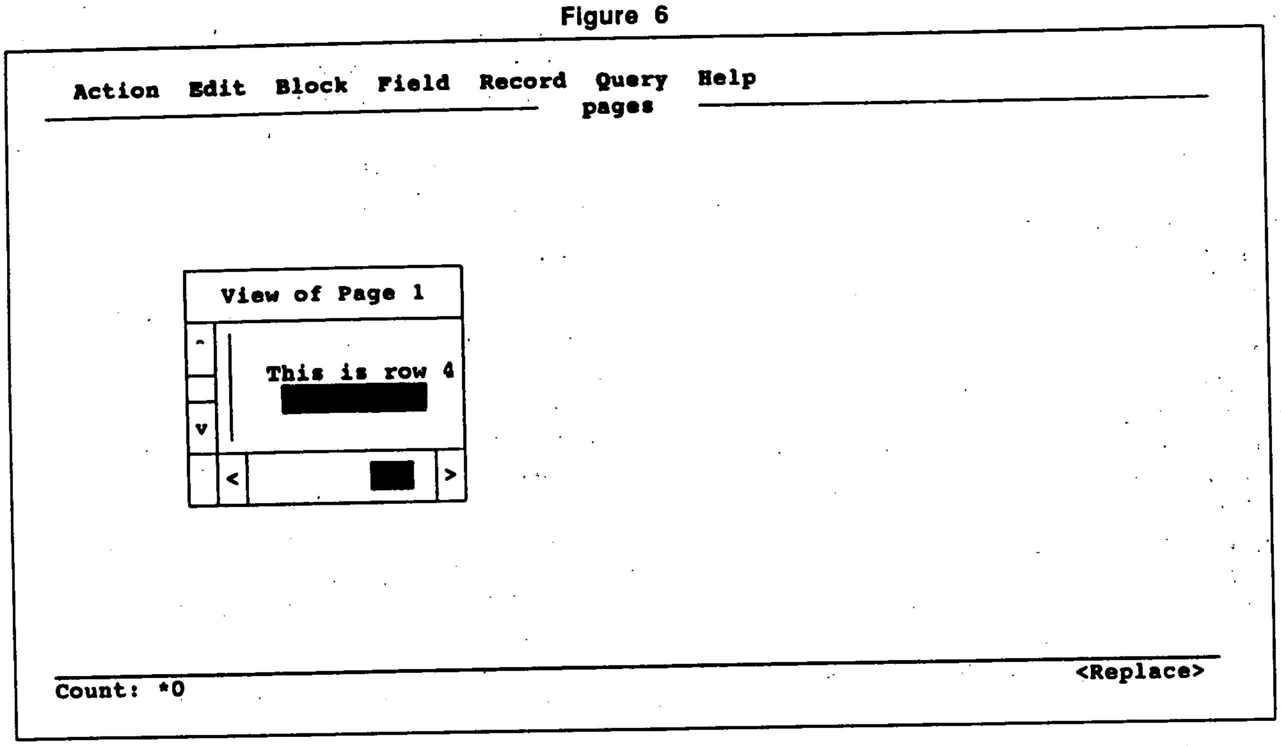
In order to show another part of this page on the screen, let's change the KEY-HELP trigger to the following:

```
ANCHOR_VIEW(1,10,8);
MOVE_VIEW(1,63,5);
```

The view can't be moved by much because it is a small view and the current field must be kept visible. When the trigger is fired, the view will be relocated on the screen and it will show a different part of the page. Figure 7 shows that the view of the page has moved down to show the text for row 7 instead of the text for row 4 and has moved to the right by 3 columns. It may be inferred that the restriction for the View Page characteristic has been violated here. The view starts at column 63 of the page and the view has been defined as 20 columns wide, so the view should extend to column 83 of the page. However, the page is defined as only 80 columns wide! This trigger works because the Border and Scroll Bar characteristics take up a total of 4 columns of the view. Only 16 columns of the page are actually shown in the view.

Let's rewrite the KEY-HELP trigger to show everything in figure 1 while keeping the border,

Figure 6



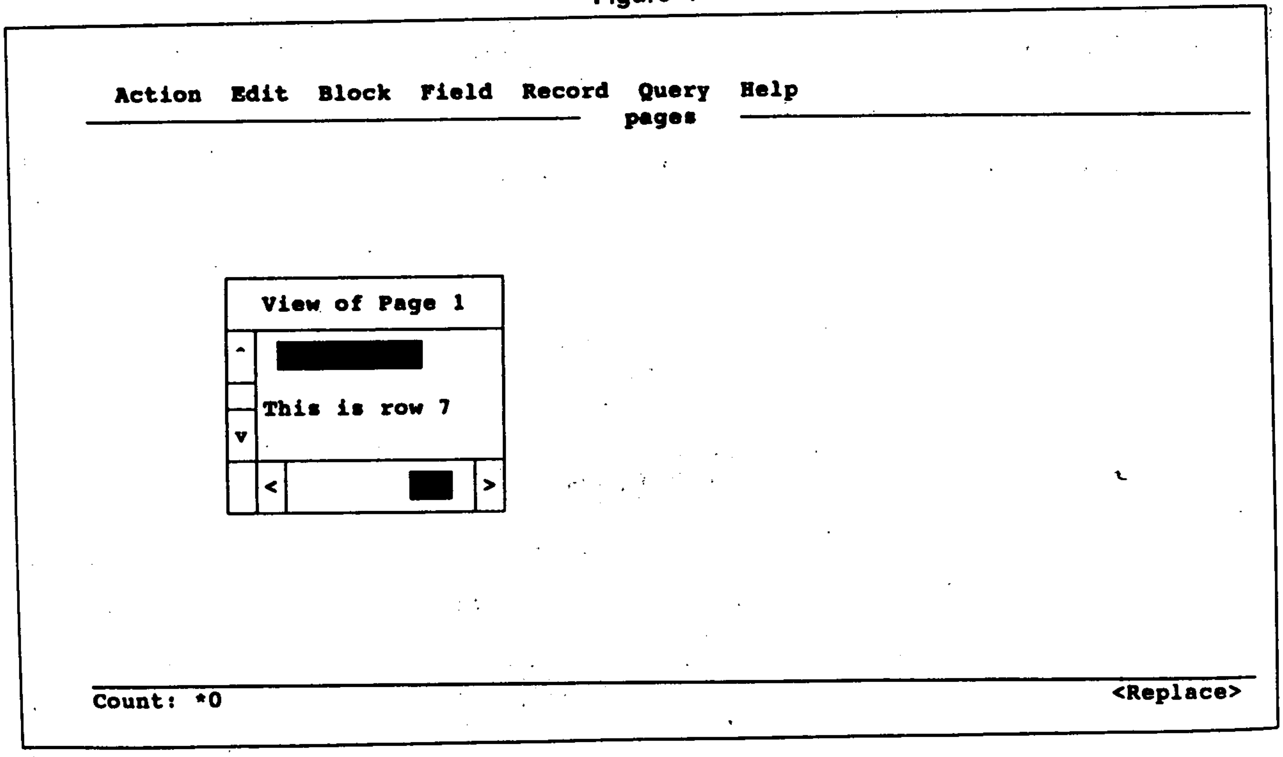
View of Page 1

This is row 4

Count: *0

<Replace>

Figure 7



View of Page 1

This is row 7

Count: *0

<Replace>

scroll bars and title. In this case, the overall view will be larger. Also, the view should be centered on the screen.

```
RESIZE_VIEW(1,24,16); -- enlarge the view
MOVE_VIEW(1,60,1);    -- move to original x,y
ANCHOR_VIEW(1,29,5);  -- center view on screen
```

Figure 8 shows the screen after this trigger has been fired. The order of these statements is important. Remember, an automatic MOVE_VIEW occurred in order to keep the field visible within the view. If the order of the MOVE_VIEW and RESIZE_VIEW were reversed in the trigger, another automatic MOVE_VIEW would occur after the trigger's MOVE_VIEW and before the RESIZE_VIEW in order to keep the current field visible within the old view size. Also, when this trigger is fired, only the final results appear on the screen. In order to see the intermediate results (the resized view and the moved view), other code would be needed. The packaged procedures, PAUSE and SYNCHRONIZE, could be used for this purpose.

To demonstrate the uses of the SHOW_PAGE and HIDE_PAGE packaged procedures, a second block and a second page have been added to the form. Figure 9 shows the page definition for this page. When the form is executed, page 1 shows up as in figure 5. Pressing [Next Block] makes a field in block two current. This field is located on page 2, so the view of page 2 is displayed on top of the view of page 1, as shown in figure 10.

Pressing [Next Block] again returns the cursor to the field which is on page 1. Page 2 disappears (instead of being overlaid) because its Remove on Exit characteristic is set. The screen will again look like figure 5. Let's again rewrite the code for the KEY-HELP trigger:

```
SHOW_PAGE(2);
PAUSE;
HIDE_PAGE(2);
```

When this trigger is fired while the cursor is in block one, SHOW_PAGE causes the view of page 2 to be displayed on the screen beneath page 1 (as shown in figure 11). The PAUSE causes the form to wait for the operator to press [Enter]. When [Enter] is pressed, HIDE_PAGE causes page 2 to disappear and the screen once again looks like figure 5. When this trigger is fired while the cursor is in block two, the only thing that happens that the operator is aware of is the PAUSE. SHOW_ PAGE has no effect because the page is already displayed and HIDE_PAGE has no effect because the current page cannot be hidden.

## More Examples

I have an application with a screen that IRS operators use to mark which tax schedules are present for a certain tax return. For any schedule which is present, they enter an 'A'. Once the schedule has been completed, this 'A' is automatically changed to a 'D'. If the operator attempts to changed the 'D' back to an 'A' (in order to redo the schedule),

Figure 8

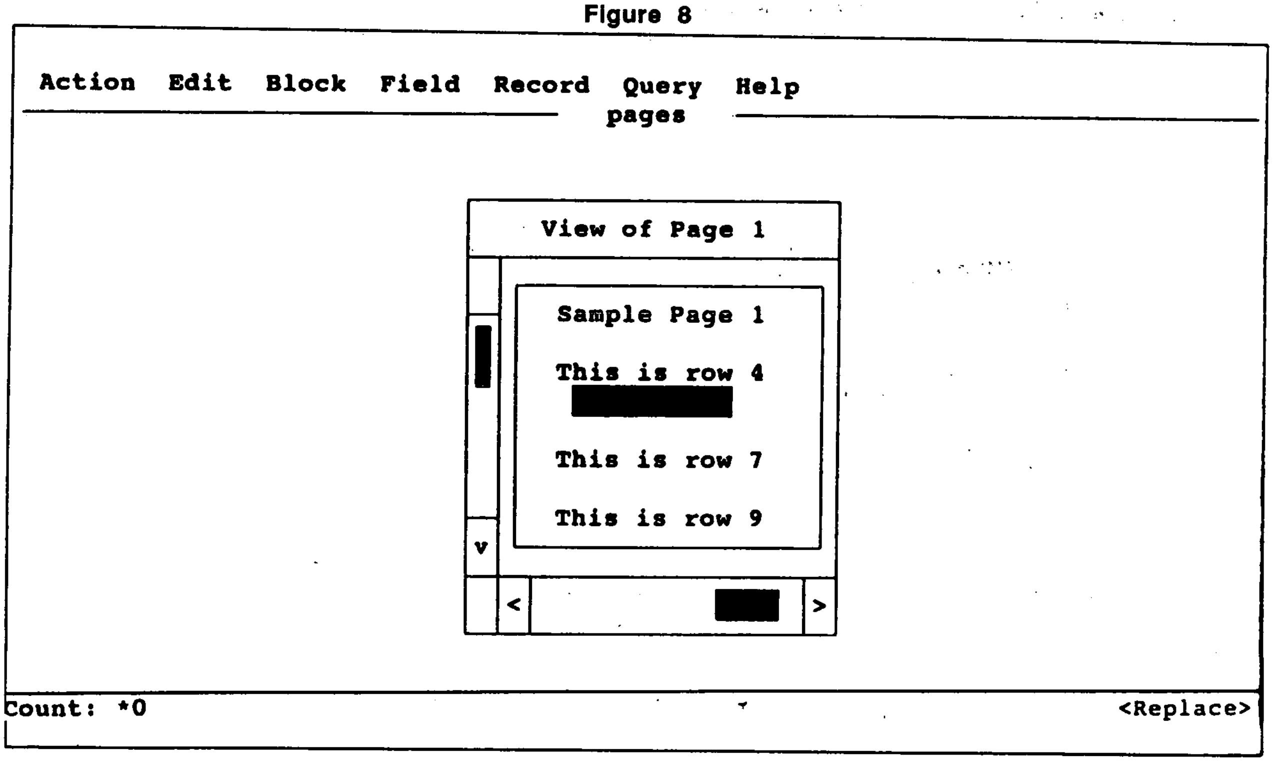
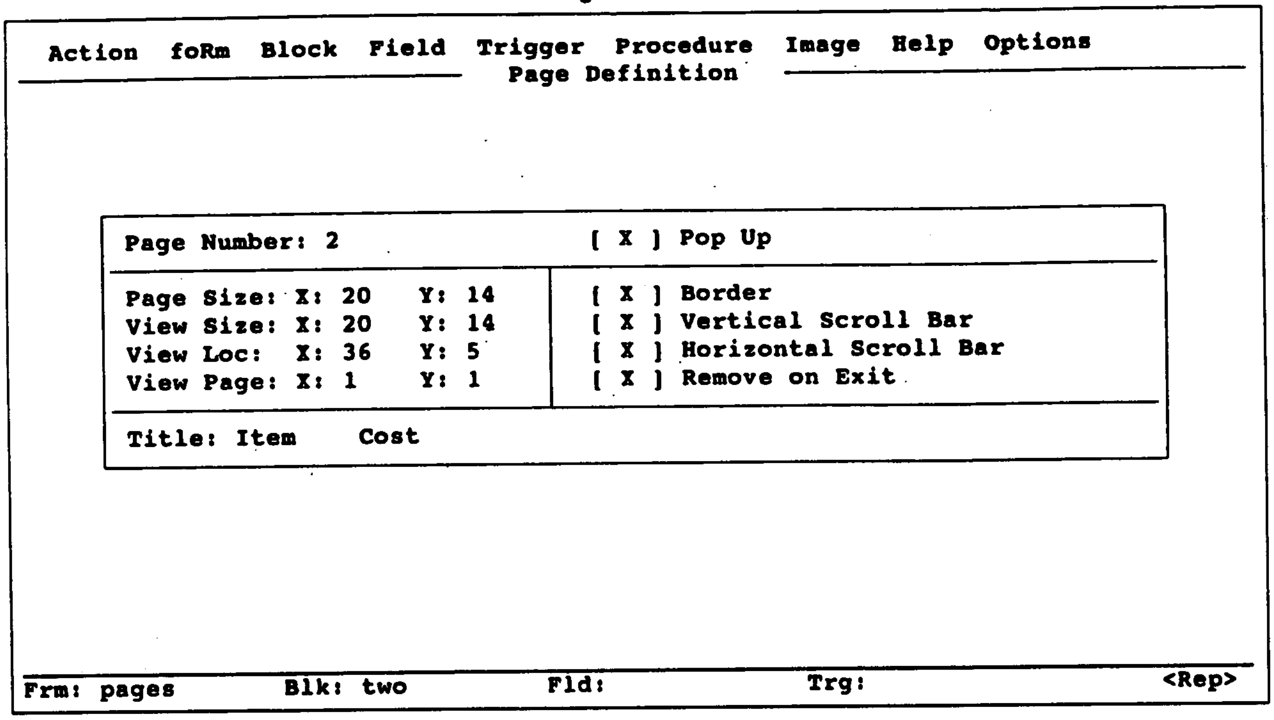


Figure 9


Action foRm Block Field Trigger Procedure Image Help Options
Page Definition

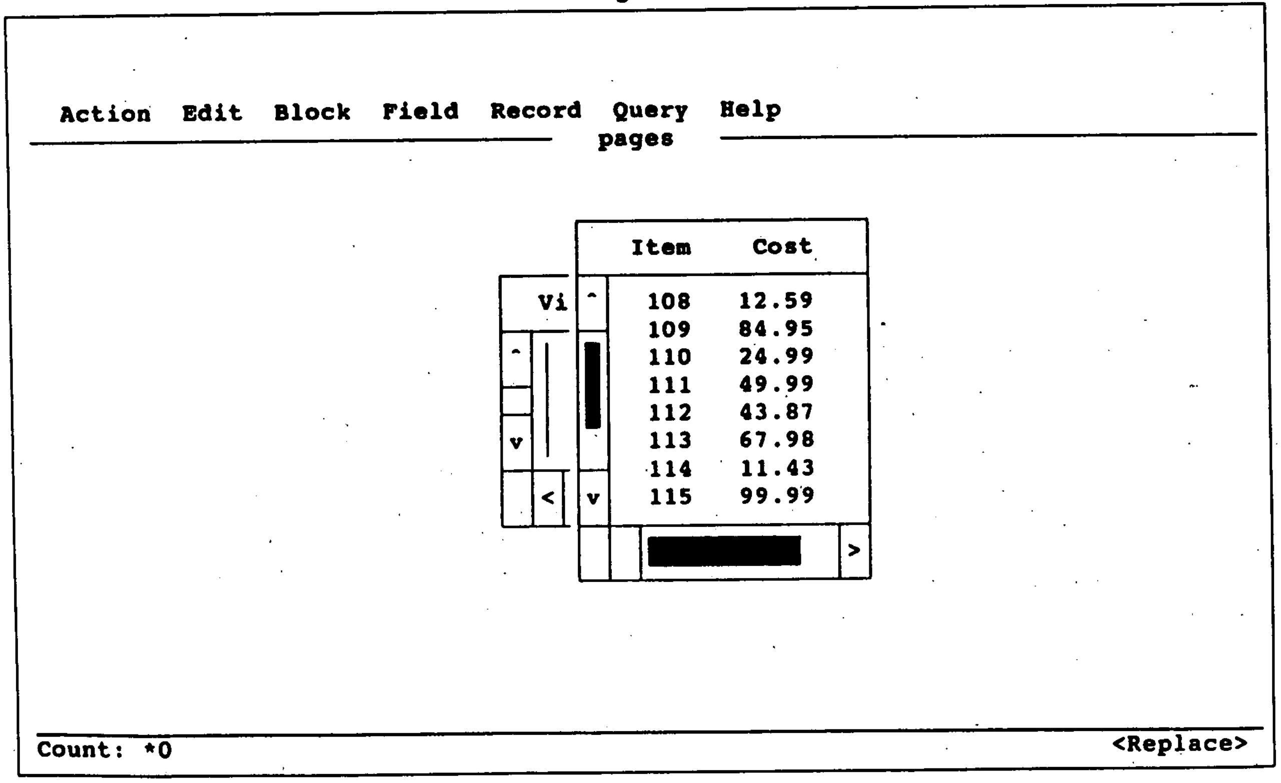
```
Page Number: 2                 [ X ] Pop Up

Page Size: X: 20   Y: 14       [ X ] Border
View Size: X: 20   Y: 14       [ X ] Vertical Scroll Bar
View Loc:  X: 36   Y: 5        [ X ] Horizontal Scroll Bar
View Page: X: 1    Y: 1        [ X ] Remove on Exit

Title: Item    Cost

Frm: pages      Blk: two      Fld:      Trg:      <Rep>
```

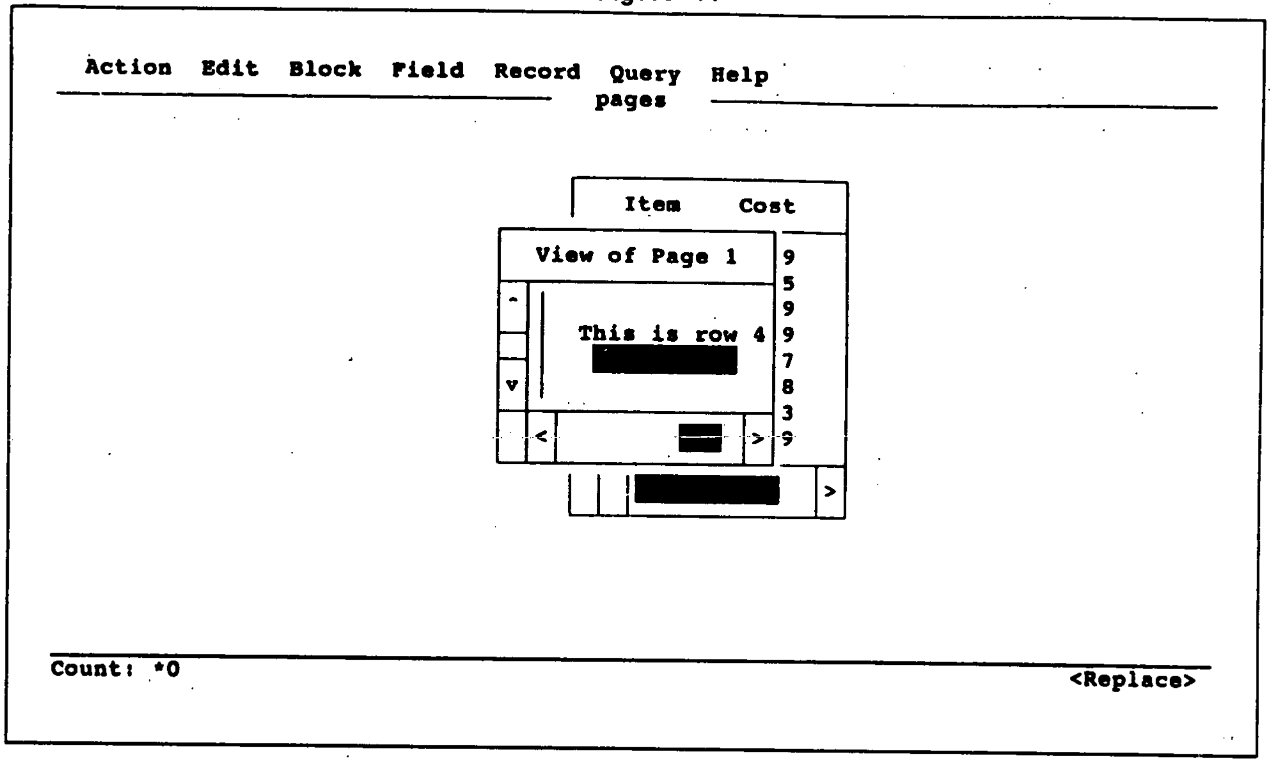
Figure 10



```
Item   Cost
108    12.59
109    84.95
110    24.99
111    49.99
112    43.87
113    67.98
114    11.43
115    99.99

Count: *0                                  <Replace>
```

Figure 11



a window pops up to confirm the change before any data are deleted. This pop up window points to the schedule in question when asking for confirmation.

Figure 12 shows the window when the operator has changed the 'D' to an 'A' for Schedule H. Figure 13 shows the same window when the operator has changed the 'D' to an 'A' for Schedule F under income type 4. The code that I used was specialized for the application but it can be generalized to the following:

```
ANCHOR_VIEW(5,

TO_NUMBER(FIELD_CHARACTERISTIC(:SYSTEM.CURSOR_FIELD,X_
POS))+2,

TO_NUMBER(FIELD_CHARACTERISTIC(:SYSTEM.CURSOR_FIELD,Y_
POS))-4);
```

The window is a view of page 5. The packaged procedure makes use of the FIELD_CHARACTERISTIC packaged function in order to find out the screen position of the current field. The window is then anchored 2 columns to the right and 4 rows above the current field. The double arrow within the window will then point at the field in question.

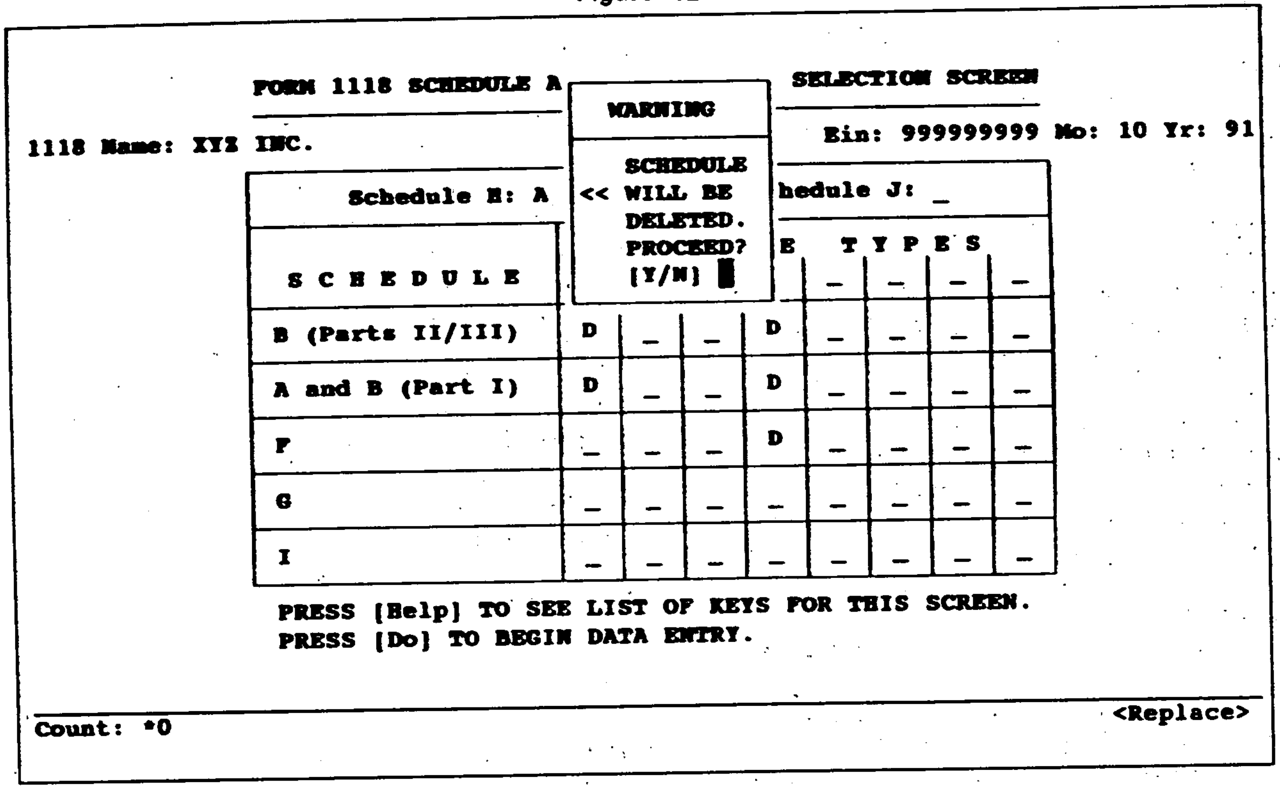
In this same application there are several pages which are used by the operators to enter the data for the various schedules. Each schedule is different, but the top two lines are always the same. They include information such as the corporation name, the employer identification number, etc. Rather than include these lines on every page (and then be forced to create and populate multiple corporate name fields, etc.), I made this header into a separate page. I then made the schedule data entry pages into pop up pages that start on row three of the screen. Once the header page is displayed, then the operator can go from schedule to schedule and the header will remain at the top of the screen.

## Fields As Text

Displaying a field as text can be useful in certain situations. For example, if a part of a screen is required to look like text but must show different information at different times, then either a separate page must be created for each variation of the text or this variable text can be made into one or more fields. The DISPLAY_FIELD packaged procedure can be used to change the way a field appears on the screen (e.g., bold, inverse video, underline or a combination of these). It should be pointed out however, that there is no display attribute which can be used to display a field as text, that is, without any special attributes.

Oracle*Terminal can be used to update the resource file in order to define a "text" attribute. The first thing that needs to be changed is the De-

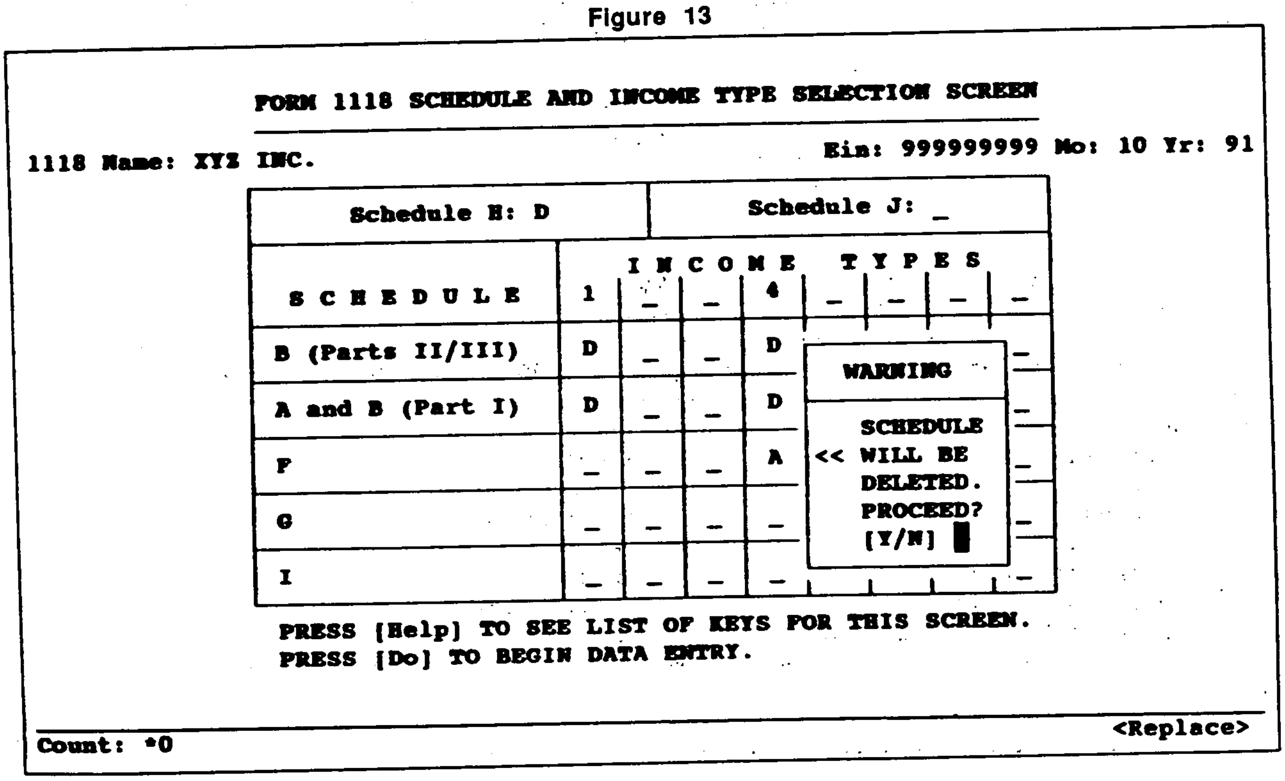
Figure 12



```
         FORM 1118 SCHEDULE A ┌──────────────┐ SELECTION SCREEN
                              │   WARNING    │
1118 Name: XYZ INC.           ├──────────────┤ Ein: 999999999 Mo: 10 Yr: 91
                              │   SCHEDULE   │
   ┌──────────────────────────│<< WILL BE    │hedule J: _          ┐
   │        Schedule H: A     │   DELETED.   │                     │
   ├──────────────────────────│   PROCEED?   │E  T Y P E S         │
   │   S C H E D U L E        │   [Y/N] █    │ _ │ _ │ _ │ _       │
   ├──────────────────────────┼───┬───┬───┬──┴┬───┬───┬───┬───┤
   │ B (Parts II/III)         │ D │ _ │ _ │ D │ _ │ _ │ _ │ _ │
   │ A and B (Part I)         │ D │ _ │ _ │ D │ _ │ _ │ _ │ _ │
   │ F                        │ _ │ _ │ _ │ D │ _ │ _ │ _ │ _ │
   │ G                        │ _ │ _ │ _ │ _ │ _ │ _ │ _ │ _ │
   │ I                        │ _ │ _ │ _ │ _ │ _ │ _ │ _ │ _ │
   └──────────────────────────┴───┴───┴───┴───┴───┴───┴───┴───┘
    PRESS [Help] TO SEE LIST OF KEYS FOR THIS SCREEN.
    PRESS [Do] TO BEGIN DATA ENTRY.

Count: *0                                                   <Replace>
```

Figure 13



```
         FORM 1118 SCHEDULE AND INCOME TYPE SELECTION SCREEN

1118 Name: XYZ INC.                          Ein: 999999999 Mo: 10 Yr: 91
   ┌──────────────────────────┬───────────────────────────────────┐
   │      Schedule H: D       │           Schedule J: _           │
   ├──────────────────────────┼───────────────────────────────────┤
   │                          │    I N C O M E   T Y P E S        │
   │   S C H E D U L E        │ 1 │ _ │ _ │ 4 │ _ │ _ │ _ │ _ │
   ├──────────────────────────┼───┼───┼───┼───┼──────────────┬───┤
   │ B (Parts II/III)         │ D │ _ │ _ │ D │   WARNING    │ _ │
   │ A and B (Part I)         │ D │ _ │ _ │ D ├──────────────┤ _ │
   │ F                        │ _ │ _ │ _ │ A │   SCHEDULE   │ _ │
   │                          │   │   │   │   │<< WILL BE    │   │
   │                          │   │   │   │   │   DELETED.   │   │
   │ G                        │ _ │ _ │ _ │ _ │   PROCEED?   │ _ │
   │                          │   │   │   │   │   [Y/N] █    │   │
   │ I                        │ _ │ _ │ _ │ _ └──────────────┘ _ │
   └──────────────────────────┴───┴───┴───┴──────────────────────┘
    PRESS [Help] TO SEE LIST OF KEYS FOR THIS SCREEN.
    PRESS [Do] TO BEGIN DATA ENTRY.

Count: *0                                                   <Replace>
```

vice Video Attribute Definition screen of Oracle*Terminal. The best solution would be to add another device video attribute name for text, but Version 1.0 only allows up to eight names. So, instead, one of the eight existing physical terminal sequences must be changed. The physical terminal sequence is the escape sequence used to set the display attributes for a field (e.g., "\e[m" means turn off bold, inverse video and underline). I changed the sequence for "inverse bold underline" because I felt it was too gaudy to be used for anything anyway. Figure 14 shows the change. This is the only change in Oracle*Terminal which needs to be made. The sequence for "inverse bold underline" is now the same as the sequence for "None". My experience is that "None" should be left alone. Many of the product attribute names are mapped to "None", so changing its sequence is not recommended. Also, trying to use a product attribute name which is mapped to "None" in order to display a field as text did not have the desired result. Instead, the result I got was that the field was displayed using the default attribute. (Note: The "product attribute name" of Oracle*Terminal is the "display attribute name" of the DISPLAY_ FIELD packaged procedure in SQL*Forms.) Now, if a trigger with the following code is fired in a form which is using the updated resource file, the field will appear with no highlighting (that is, as text):

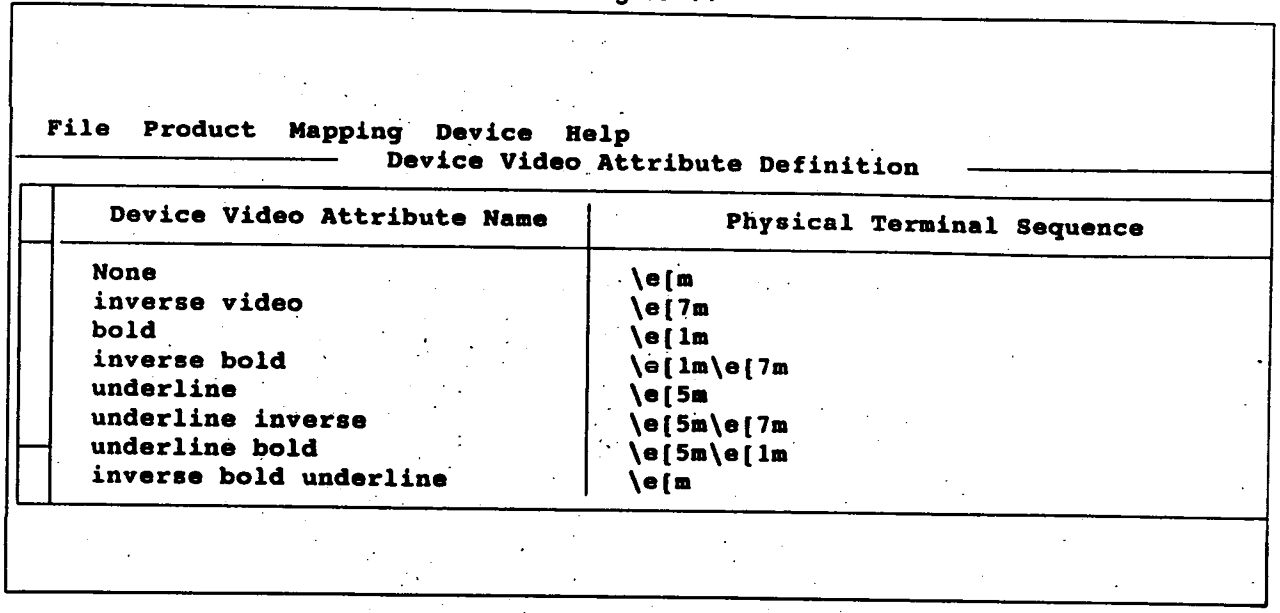
```
DISPLAY_FIELD(text_field_name,'BOLD-INVERSE-UNDERLINE'
);
```

However, this code would be confusing to anyone who inherited the maintenance responsibilities for the form. In order to make the code more self-documenting, the display attribute name can be changed. Back in Oracle*Terminal, the Product Attribute Definition screen is called up and a new product attribute name, Text, is inserted into the list and given a description of "Text". This step adds a new name to the list of valid product attribute names. See figure 15. Next, the Attribute Mapping Definition screen is updated, as in figure 16, to include the product attribute name, Text. This name is then mapped to the device video attribute name, inverse bold underline. This step links the new product attribute name, Text, to the physical terminal sequence, \e[m. The trigger code can now be changed to:

```
DISPLAY_FIELD(text_field_name,'TEXT');
```

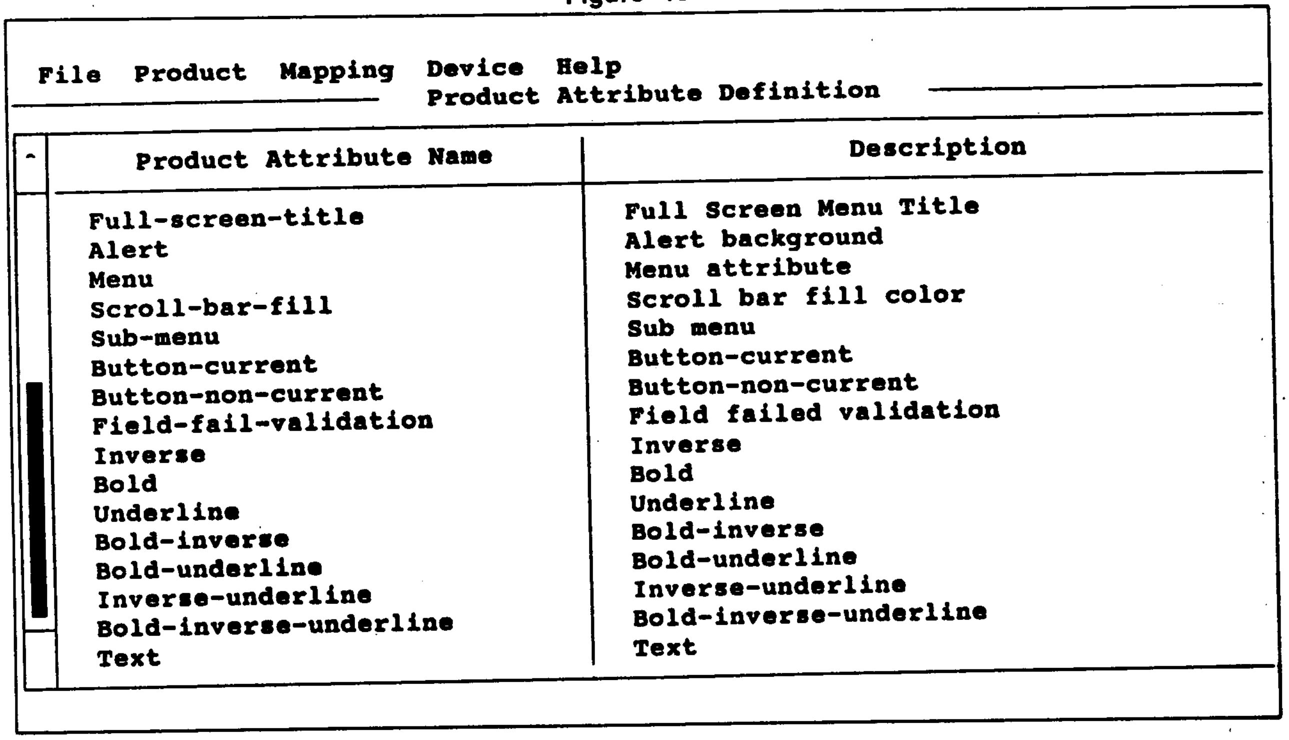
Warning: If the block containing the field is cleared via CLEAR_BLOCK, the record containing the field is cleared via CLEAR_RECORD or the form is cleared via CLEAR_FORM, then the field will revert back to its default display attributes. (This is not mentioned in the documenta-

Figure 14


File Product Mapping Device Help
Device Video Attribute Definition

| Device Video Attribute Name | Physical Terminal Sequence |
|---|---|
| None | \e[m |
| inverse video | \e[7m |
| bold | \e[1m |
| inverse bold | \e[1m\e[7m |
| underline | \e[5m |
| underline inverse | \e[5m\e[7m |
| underline bold | \e[5m\e[1m |
| inverse bold underline | \e[m |

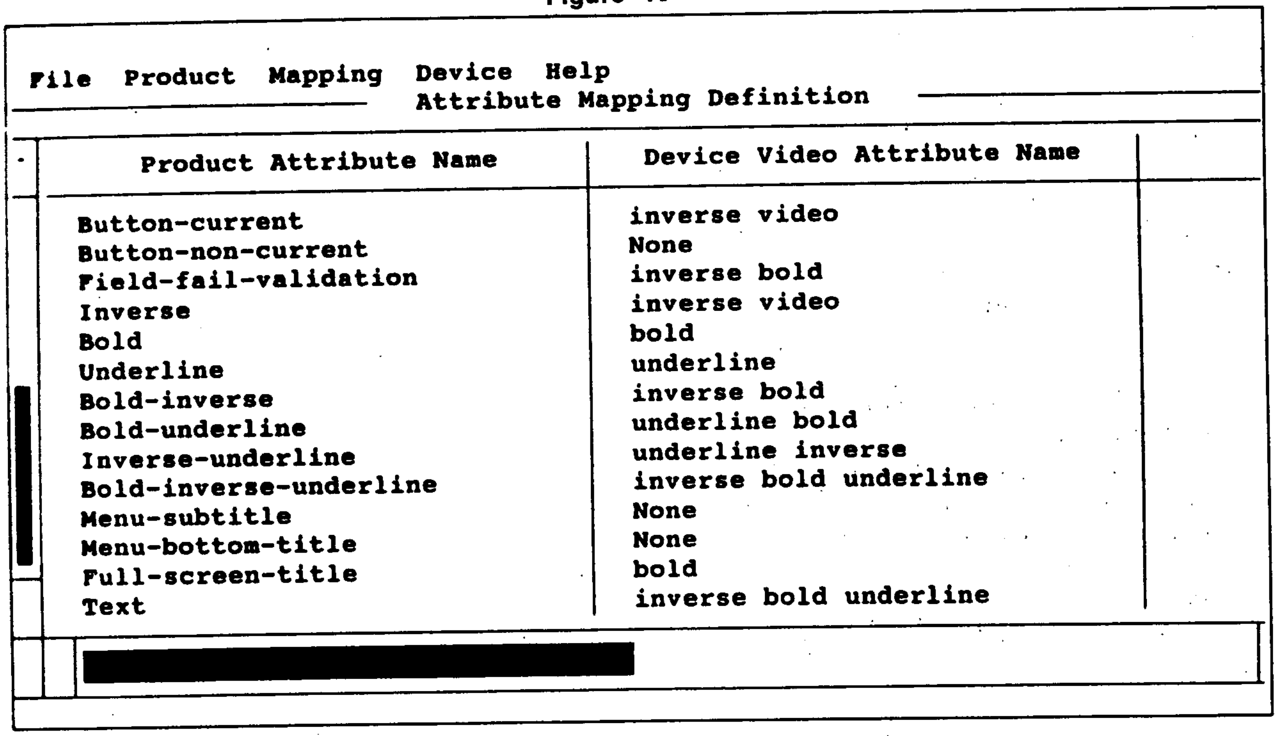
Figure 15



Product Attribute Definition

| Product Attribute Name | Description |
|---|---|
| Full-screen-title | Full Screen Menu Title |
| Alert | Alert background |
| Menu | Menu attribute |
| Scroll-bar-fill | Scroll bar fill color |
| Sub-menu | Sub menu |
| Button-current | Button-current |
| Button-non-current | Button-non-current |
| Field-fail-validation | Field failed validation |
| Inverse | Inverse |
| Bold | Bold |
| Underline | Underline |
| Bold-inverse | Bold-inverse |
| Bold-underline | Bold-underline |
| Inverse-underline | Inverse-underline |
| Bold-inverse-underline | Bold-inverse-underline |
| Text | Text |

Figure 16



Attribute Mapping Definition

| Product Attribute Name | Device Video Attribute Name |
|---|---|
| Button-current | inverse video |
| Button-non-current | None |
| Field-fail-validation | inverse bold |
| Inverse | inverse video |
| Bold | bold |
| Underline | underline |
| Bold-inverse | inverse bold |
| Bold-underline | underline bold |
| Inverse-underline | underline inverse |
| Bold-inverse-underline | inverse bold underline |
| Menu-subtitle | None |
| Menu-bottom-title | None |
| Full-screen-title | bold |
| Text | inverse bold underline |

tion for the DISPLAY_FIELD packaged procedure.)

## Summary

The purpose of this paper was to share my experiences in SQL*Forms 3.0 with the reader in the hope that he or she will travel a shorter path to becoming proficient at developing forms.

Many of the characteristics of a page can be customized in the Page Definition Table. These characteristics can then be further modified as needed via packaged procedures. From the examples shown here, it can be seen that the pop up page is a very useful tool. These examples are just a beginning; much more can be done with pop up pages.