Overview

Use the Text Entry class to create text fields that accept user input and display information or program output. These text fields can contain up to 200 characters of text and can contain one or more lines.

PARENT: SASHELP.FSP.WIDGET.CLASS
CLASS: sashelp.fsp.EField.class

Using the Text Entry Class

SCL programs are easier to maintain if you do not code numbers for colors or display attributes in the programs. If color or display attribute numbers were to change, your programs could produce unexpected results.

To make your programs easier to maintain, you can create a field, assign it the appropriate validation type COLOR or ATTR, and assign the appropriate value to the field. This technique enables you to use the name of this text object for the argument that specifies the appropriate parameter number. The objects to which you send color or display attribute changes will always display correctly, regardless of the internal number.

For example, if you know that the number assigned to the color red is 2, you can use the following statement to display the field TEXT1 in red:

```scl
call notify('text1','_set_color_num_',2);
```

However, if the internal numbers assigned to colors were to change, the next time you run your SCL program the text would display in whatever color was assigned the number 2.

Alternatively, you can construct a statement that generates an error message and halts the program if the number 2 was no longer assigned to a color. Better still, if you create a field with the validation type COLOR and the value `red`, as shown in the next example with COLORNUM, you can use the following statement and know that the correct color will be assigned, regardless of color number assignments:
Another way to do this is to use the supplied color or display attributes lists (stored in SASHELP.FSP.COLORS.SLIST and SASHELP.FSP.ATTRS.SLIST). The following example prints the lists and gets the numbers associated with the color red and the attribute reverse and applies them to the text entry object, OBJ2:

```
INIT:
  colors=makelist(); /* create list */
  attrs=makelist();

  fillist('catalog',
    'sashelp.fsp.colors.slist',
    colors); /* fill it */
  fillist('catalog',
    'sashelp.fsp.attrs.slist'
    ,attrs);

  call putlist(colors,'colors',1);
    /* print it */
  call putlist(attrs,'colors',1);

  colornum=getnitemn(colors,'red');
    /* get info */
  attrnum=getnitemn(attrs,'reverse');

  call notify ('obj2','_set_color_num_',
    colornum,attrnum);
    /* apply it */
return;
```

**Methods**

Methods specific to the Text Entry class are described here. Inherited methods are described in the Object class and the Widget class.

**Dictionary**

**_getCurcol**

Returns the position of the text cursor relative to the first column of a text entry field.

**Syntax**

```
CALL NOTIFY ('text-field-name', _getCurcol', column);
```
### _getCurcol

_**getCurcol** method returns the position of the text cursor (relative to the text entry field) when a user either clicks with the mouse or presses the RETURN or ENTER key. The cursor can be in the master region or in another object. If the text cursor is in an area to the left of the field’s first column, the number returned is less than or equal to zero; if the cursor is in an area to the right of the first column, the number returned is positive.

#### Examples

_**getCurcol** returns the cursor position relative to the first column of the field TEXT1. TEXT1 has a length of 6.

- Assume the cursor is on the third character of TEXT1, as shown here:
  
<table>
<thead>
<tr>
<th>X</th>
<th>cursor position</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6</td>
</tr>
</tbody>
</table>
  
  This statement returns a 3 in the SCL variable MYCOL:

  ```
  call notify ('text1','_get_curcol_','mycol');
  ```

- Assume the cursor is 5 columns (in any row) to the left of the first column of TEXT1, as shown here:

<table>
<thead>
<tr>
<th>X</th>
<th>cursor position</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6</td>
</tr>
</tbody>
</table>
  
  This statement returns a -4 in the SCL variable MYCOL:

  ```
  call notify ('text1','_get_curcol_','mycol');
  ```

### _getCurrow

_**getCurrow** method returns the position of the text cursor relative to the first row of a text entry field.

#### Syntax

```
CALL NOTIFY (text-field-name, '_getCurrow', row);
```
### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>N</td>
<td>returns the cursor position relative to the field’s first row:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;0 the number of rows below the field’s first row plus 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;=0 the absolute number of rows above the field’s first row minus 1</td>
</tr>
</tbody>
</table>

### Details

The `_getCurrow` method returns the position of the text cursor (relative to the text entry field) when a user clicks with the mouse or presses the RETURN or ENTER key. The cursor can be in the master region or in another object. If the cursor is in an area below that first row, the number returned is positive. If the cursor is in an area above the receiver field’s first row, the number returned is less than or equal to zero.

### Examples

 `_getCurrow` returns the cursor position relative to the first row of the field TEXT1. TEXT1 has a length of 6.

1. Assume a user presses the ENTER key when the cursor is on the first row of TEXT1, as shown here:

<table>
<thead>
<tr>
<th>row number</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
</tr>
<tr>
<td>-1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>X_ _ _ _</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

   (cursor on row 1 relative to position of TEXT1 field)

   This statement returns a 1 in the SCL variable MYROW:

   ```
call notify ('text1','_get_currow_','myrow');
```

2. Assume the user presses the ENTER key when the cursor is 3 rows (in any column) above TEXT1, as shown here:

<table>
<thead>
<tr>
<th>row number</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>-1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>___ ___ ___ (field position)</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

   This statement returns a -2 in the SCL variable MYROW:

   ```
call notify ('text1','_get_currow_','myrow');
```
_getText

Returns the text of a text entry field

Syntax

CALL NOTIFY (text-field-name, '_getText', string);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>C</td>
<td>returns the contents of the field</td>
</tr>
</tbody>
</table>

Details

The _getText method is inherited from the Widget class. It returns the contents of any field including the numeric types NUMLST and FIXEDLST, which can contain strings of numbers.

Note: It is possible to obtain unexpected results between successive _setText and _getText statements. For example, suppose you sent a _setText method with the value Jelly to a field named TEXT1 that had the validation type ACTION. The next _getText method sent returns the value X and not Jelly. Another problem can occur when you send text in mixed case to a field with the CAPS attribute turned on, and the text displays in uppercase letters. Subtle differences like this can cause programming errors.

Instead of using this method, you can also use simple SCL statements that assign text to a field or return text stored in a field. For example, if field TEXT1 contains the string hello, the following PUT statement displays TEXT1=hello and the next statement assigns hello to the SCL variable A:

```plaintext
put text1=;
A = text1;
```

However, SCL statements and functions do not work across Frame boundaries. Therefore, if you need to access field contents between two or more Frames, use the appropriate method instead.

Examples

- In this example, suppose the text value 25 45 82 is stored in the field TEXT1, which has the type FIXEDLST. The following statement assigns this value to the variable MYSTR:

  ```plaintext
call notify ('text1','_get_text_','mystr);
```

- In this example, if the field TEXT2 contains the text jelly rolls, the following statement returns that value and assigns it to the variable MYSTR:

  ```plaintext
call notify ('text2','_get_text_','mystr');
```
**_getValue**

*Returns the value of a simple numeric text entry field of type NUM, FIXED, or SHORT*

**Syntax**

CALL NOTIFY (text-field-name, _getValue_, text-value);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text-value</td>
<td>N</td>
<td>returns the field's value</td>
</tr>
</tbody>
</table>

**Details**

_getValue is inherited from Widget. The potential for programming errors that exists with successive _setText and _getText statements also applies to _setValue and _getValue. For more information, see _getText in this class.

**Example**

_getValue puts the value of the field TEXT1 into the numeric SCL variable NVAL:

    call notify ('text1','_get_value_',nval);

---

**_getType**

*Returns type information about a text entry field*

**Syntax**

CALL NOTIFY (text-field-name, _getType_, type, specific);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>C</td>
<td>returns the type value of the field:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CHAR' character</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'NUM' numeric</td>
</tr>
<tr>
<td>specific</td>
<td>C</td>
<td>a specific type value.</td>
</tr>
</tbody>
</table>

**Details**

_getText is inherited from Widget.
_setAutoflow

Specifies whether to flow the text in the text entry field

Syntax

CALL NOTIFY (text-field-name, '_setAutoflow', action);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>C</td>
<td>specifies the setting:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘Y’ flows the text without regard to spacing or word breaks. Multiple blanks are reduced to one blank (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘N’ leaves all text as it was entered</td>
</tr>
</tbody>
</table>

_setAutoskip

Specifies whether to move the cursor to the next unprotected object when input fills the text entry’s last input column

Syntax

CALL NOTIFY (text-field-name, '_setAutoskip', action);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>C</td>
<td>specifies the setting:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘Y’ moves the cursor to the next unprotected object when the last column of the text entry is reached (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘N’ leaves the cursor on the last input column of the text entry</td>
</tr>
</tbody>
</table>

_setCaps

Specifies whether to make uppercase any characters entered in the text entry field
Syntax

**CALL NOTIFY** (text-field-name, '_setCaps', action);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>C</td>
<td>specifies the setting:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' makes characters in the field uppercase when the field is verified (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' leaves characters as entered</td>
</tr>
</tbody>
</table>

_setColorName_

Assigns the text entry field color and/or display attribute specified by name

Syntax

**CALL NOTIFY** (text-field-name, '_setColorName', color, <attribute>, start-column, length>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>C</td>
<td>assigns the name of a color.</td>
</tr>
<tr>
<td>attribute</td>
<td>C</td>
<td>assigns the name of a display attribute.</td>
</tr>
<tr>
<td>start-column</td>
<td>N</td>
<td>specifies the column number where the changes start</td>
</tr>
<tr>
<td>length</td>
<td>N</td>
<td>specifies the number of columns to be affected by the changes</td>
</tr>
</tbody>
</table>

Examples

- This example displays blue text in the field TEXT1:
  ```
  call notify ('text1', '_set_color_name', 'blue');
  ```
- This example displays reverse video text in the field TEXT1. Because this statement specifies only the display attribute, a null value serves as a placeholder for the color argument, and the color remains the same.
  ```
  call notify ('text1', '_set_color_name', '', 'reverse');
  ```
- This example colors five characters green in the field TEXT1, beginning at column 4. Because this statement does not specify a display attribute, a null value is used for the display attribute argument, and the attribute remains the same.
  ```
  call notify ('text1', '_set_color_name', '', 4, 5, 'green');
  ```
call notify ('text1','_set_color_name_','green','_',4,5);

- This example changes the style of four characters in the text to red, reverse video, starting at column 1:
  
call notify ('text1','_set_color_name_','red','reverse',1,4);

=setColorNum

Assigns the field color and/or display attribute specified by number

Syntax

CALL NOTIFY (text-field-name, '_setColorNum', color, <attribute>, start-column, length);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>N</td>
<td>assigns to the object the value of type COLOR. See the refid=commonchap page=no chapter.</td>
</tr>
<tr>
<td>attribute</td>
<td>N</td>
<td>assigns to the object the value of type ATTR.</td>
</tr>
<tr>
<td>start-column</td>
<td>N</td>
<td>specifies the column number where the changes start</td>
</tr>
<tr>
<td>length</td>
<td>N</td>
<td>specifies the number of columns to be affected by the changes</td>
</tr>
</tbody>
</table>

Details

See “Using the Text Entry Class” on page 1891.

Examples

- In this example, change the color for the field TEXT1 to the value of the field COLORNUM. COLORNUM is a numeric field with the type COLOR that has been assigned the value blue.
  
call notify ('text1','_set_color_num_','colornum');

- In this example, change the display attribute for TEXT1 to the value of the field ATTRNUM. ATTRNUM is a numeric field with the type ATTR, whose value is highlight. In this statement, a null placeholder is used for the color argument because only the text attribute is being changed:
  
call notify ('text1','_set_color_num_','. ,attrnum');

- In this example, assign the color stored in the field COLORNUM to five characters of the field TEXT1, starting at column 4. COLORNUM is a numeric field with the
Chapter 102

A type COLOR whose value is green. Again, note the null placeholder for the display attribute argument.

call notify ('text1', '_setColor_num_',
    colornum, , 4, 5);

In this example, use object variables to assign a color and display attribute to the first four characters in the field TEXT1. The color is stored in the field COLORNUM, a numeric field with the type COLOR whose value is red. The display attribute is stored in the field ATTRNUM, a numeric field with the type ATTR, whose value is reverse.

call notify ('text1', '_setColor_num_',
    colornum, attrnum, 1, 4);

SetColorStr

Assigns the text entry field color and/or display attribute specified by an SCL attribute string

Syntax

CALL NOTIFY (text-field-name, '_setColorStr', attribute-string);

Argument Type Description

| attribute-string   | C     | assigns the SCL attribute string containing the color and highlighting specifications. |

Details

This method enables you to use the STRATTR function by creating an SCL attribute string and assigning its value with the STRATTR function.

Example

This example uses the STRATTR function to create an SCL attribute string, STR, containing the values red and reverse. _setColorStr uses STR to change the text color and display attribute for TEXT1 to red and reverse, respectively.

str=strattr('red', 'reverse', 1, mlength(str));
call notify('text1', '_setColor_str_', str);

SetColorJustify

Specifies the alignment of the text in a text entry field
Syntax

CALL NOTIFY (text-field-name, '_setNoprompt', justify);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>justify</td>
<td>C</td>
<td>specifies a justification value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT' (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CENTER'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'NONE'</td>
</tr>
</tbody>
</table>

_setNondisplay

Specifies whether to display the text that is entered in the field

Syntax

CALL NOTIFY (text-field-name, '_setNondisplay', action);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>C</td>
<td>specifies the setting:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' hides the text that is entered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' echoes the text that is entered</td>
</tr>
</tbody>
</table>

Details

If the NON-DISPLAY attribute is set to 'Y', text is not displayed. Although users can tab to the field, user input is not echoed to the display. This is often used for entering passwords. By default, the NON-DISPLAY attribute is off.

_setNoprompt

Specifies whether to honor special prompt character processing

Syntax

CALL NOTIFY (text-field-name, '_setNoprompt', action);
Argument Type Description

<table>
<thead>
<tr>
<th>action</th>
<th>C</th>
<th>specifies the setting:</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Y'</td>
<td></td>
<td>honors special prompt processing</td>
</tr>
<tr>
<td>'N'</td>
<td></td>
<td>ignores special prompt processing</td>
</tr>
</tbody>
</table>

**Details**

If `_setNoprompt` is 'Y', entering the default prompt character, a question mark (?), does not display information about the text object. The prompt character is passed instead to the SCL program for processing. By default, NOPROMPT is off.

---

**_setRequired**

Specifies whether the text entry field must contain a valid value

---

**Syntax**

CALL NOTIFY (text-field-name, '_setRequired', action);

---

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>C</td>
<td>specifies the setting:</td>
</tr>
<tr>
<td>'Y'</td>
<td></td>
<td>a valid value is required before the window can close with an END command</td>
</tr>
<tr>
<td>'N'</td>
<td></td>
<td>a valid value is not required (the default)</td>
</tr>
</tbody>
</table>

---

**_setText**

Assigns text to a text entry field

---

**Syntax**

CALL NOTIFY (text-field-name, '_setText', text-value);
**_setValue**

Assigns a value to a simple numeric text entry field of type NUM, FIXED, or SHORT

**Syntax**

```
CALL NOTIFY (text-field-name, '_setValue', text-value);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text-value</td>
<td>N</td>
<td>specifies the numeric value to assign to the field</td>
</tr>
</tbody>
</table>

**Details**

_setValue is inherited from Widget.

---

**_setText** is inherited from Widget. Character-based fields include the numeric types NUMLST and FIXEDLST, which can contain strings of numbers.

**Examples**

- This example assigns the text `25.5 45 82` to the field TEXT1, which is a field with the type NUMLST.
- This example assigns the text `jelly rolls` to COLORNUM, a field with the type CHAR.

```scl
  call notify ('text2', '_set_text', 'jelly rolls');
```

You can also use simple SCL assignment statements to set text values.

```scl
  text1 = 'my text';
  text2 = mystr;
```

However, remember that SCL statements and functions do not work across FRAME entry boundaries. So, if you need to set text values between two or more FRAME entries, use the _setText method instead.

See _getText in this class for warnings about potential programming errors that also apply to this method.
Example

This example assigns the value 25 to TEXT1, a field with the type FIXED:

```call notify ('text1','_set_value_','25');
```

For more information, see the note in _getText in this class.

---

_validate

Provides a model for a user-defined _validate method to be used with a subclass of Text Entry

Syntax

CALLSUPER(_SELF_, '_validate');

Details

_validate is not valid unless you have subclassed a text entry field and overridden the supplied _validate method with your own. _validate is called automatically when a field is modified.

Example

_validate adds customized validation to the default validation:

```VALIDATE:
method;
   /* turn off error flags */
   call send(_self_,'_erroroff_');
   /* do supplied validation */
   call super(_self_,'_validate_');
   /* get field value */
   call send(_self_,'_get_value_','nval');
   /* do custom validation */
   if (nval>goodval) then
      call super(_self_,'erroron_');
endmethod;
```