Introduction to JYACC FORMAKER

Contents

Contents

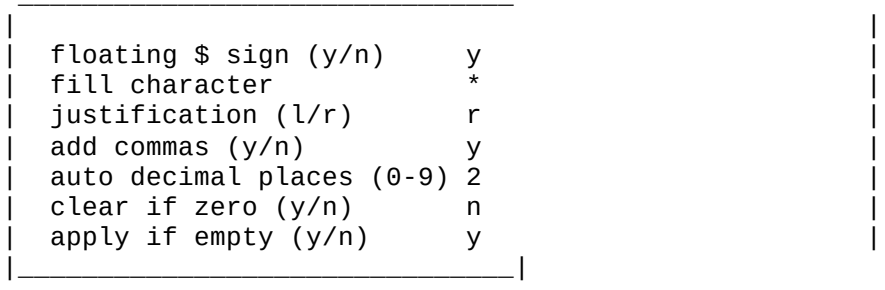

Figure 30: Amount Edits Window

The field is validated when the last position of the field is filled, or when the operator attempts to leave the field by use of the tab or new line (return) key. Non-numeric characters within the field are ignored.

5.5.5.13 Amount Fields

An "amount field" is a numeric field in which any data entered are formatted, as specified, after you tab out of the field. During formatting, the amount is right justified, unless you specify left justification. To set or reset amount edits:

1. Bring up the special edits menu (Section 7.5.5).
2. Choose "amount edits." The amount edits window will be brought up, with the currently set amount edits, if any, displayed (Figure 30).
3. Enter or erase the desired data. (Use ERASE ALL UNPROTECTED to remove the amount edits altogether.)
4. Hit TRANSMIT to save the edits (or EXIT to abort).

The following options are available:

floating $ sign	Enter "y" to insert a floating dollar sign, "n" or no entry to omit it. (For a fixed dollar sign, omit this option. Instead, enter a $ as display data on the form, in the last character position preceding the field.)
fill character	Enter a fill character, if desired, or leave blank. The character specified will replace any blanks in the field.
justification	Enter l for left justification, r or no entry for right justification.
add commas	Enter y to insert commas, n or no entry to omit them.

NAME

amt_format - write data from a buffer to a field, applying amount field editing as appropriate

SYNOPSIS

```
sm_amt_format (field_number, buffer);
int field_number;
char *buffer;
```

DESCRIPTION

If the specified field has an amount field edit attached, the edit is first applied to the data in the buffer. If the resulting string is too long for the field, an error message is displayed. Otherwise, putfield is called to write the edited string to the specified field.

If the specified field has no amount edit, putfield is called with the unedited string.

RETURNS

0 is returned if the field is updated successfully. -1 is returned if the named field does not exist, or if the field number or item ID is out of range. -2 is returned if the formatted string is too long for the field.

VARIANTS AND RELATED FUNCTIONS

```
sm_o_amt_format (field_number, occurrence, buffer)
sm_n_amt_format (field_name, buffer)
sm_i_amt_format (field_name, occurrence, buffer)
sm_e_amt_format (field_name, element, buffer)
```

NAME

bel - beep!

SYNOPSIS

```
sm_bel ();
```

DESCRIPTION

Causes the display to beep, usually (but not always) by transmitting the ASCII BEL code to it.

This function should be used in place of putchar(7) or putchar(BEL), because certain displays use that as a graphics character.

NAME

choice - get item selection

SYNOPSIS

```
text = sm_choice (type);
char *text;
int type;
```

DESCRIPTION

This is a menu-handling function, similar in some respects to menu_proc. It enables you to tab, backtab, arrow and scroll through a screen, in order to select the contents of one of the fields or scrolling items. The entry at which the cursor is positioned is shown in reverse video.

Hitting a key that matches the first character of a screen entry causes the cursor to be positioned there; if more than one entry begins with that character, the cursor is positioned to the first entry following its current location. Entries are searched by field number. Arrays and scrolls, however, are searched in their entirety following their first field, and scrolling occurs automatically.

If "type" is UPPER (or LOWER), an alphabetic key is translated to upper (or lower) case before a match is attempted; if "type" has any other value, the entry is not translated. choice returns to the calling program only when you hit the TRANSMIT or EXIT key.

Note that mp_options and mp_string, which control the behavior of menu_proc, do not affect choice. Also unlike menu_proc, this function ignores the RETURN ENTRY (or MENU) edit.

RETURNS

```
text = pointer to text of the selected field if the TRANSMIT key was hit.
     = 0 if the EXIT key was hit.
```

VARIANTS AND RELATED FUNCTIONS

```
sm_menu_proc (type);
```

NAME

d_at_cur - display memory-resident window at current cursor position

SYNOPSIS

```
sm_d_at_cur (form_ptr);
char *form_ptr;
```

DESCRIPTION

Displays a memory-resident screen as a window with its upper left-hand corner at the current cursor position (offset one line to avoid hiding that line's current display). If the window will not fit there, the starting position is automatically adjusted. The argument 'form_ptr' is the address of a character array containing the screen data; the code for such an array may be generated automatically using the FORM2ASC utility.

This function is similar to r_at_cur, except that the screen is in memory; this routine bears the same relationship to r_at_cur as d_form bears to r_form. The memory-resident form is never altered, and may therefore be made shareable on systems that support shareable read-only sections.

RETURNS

0 is returned if no error occurred during display of the form. If the screen is too big for the physical display but no fields need be truncated, this routine displays a warning message and as much of the screen as will fit, and returns 0. -3 is returned if no memory was available, or if the screen contained fields that would not fit within the physical display. The screen is always restored to its previous condition.

NAME

d_form - display memory resident form

SYNOPSIS

```
sm_d_form (form_ptr);
char *form_ptr;
```

DESCRIPTION

This function displays a memory-resident screen as a base form. Any forms and windows that are currently displayed are lost, and their memory is released. The argument 'form_ptr' is a pointer to the character array holding the screen data. The FORM2ASC utility can create a source module containing such arrays from the screen data files; that module can then be compiled and linked with the application.

The action of this function is similar to that of r_form. The form array itself is never modified; under certain systems it is therefore possible to arrange that the forms be placed in a shareable section.

If this routine returns an error, any previously displayed screen is lost, and the current form is undefined.

RETURNS

0 is returned if no error occurred during display of the form. Warning messages are issued, with a 0 return, if the screen is larger that the physical display but no fields need be truncated. -5 is returned if, after the screen was cleared, the system ran out of memory. -7 is returned if the screen contained fields that would not fit within the physical display.

NAME

d_msg_line - display message on status line

SYNOPSIS

```
sm_d_msg_line (message_text, initial_attribute);
char *message_text;
char initial_attribute;
```

DESCRIPTION

If the 'message_text' pointer passed does not already point to the status line buffer, the message is copied into the status line buffer. The message in the status line buffer is then displayed on the physical status line, with the given initial display attribute.

If the hex character 0x80 is encountered within the message text, the following character is taken to be an attribute, and the display attribute is reset at that point. The attribute character is interpreted as the sum of one or more of the following:

Attribute	Defined value (hexadecimal)
	NORMAL_ATTR 0x07
	BLACK 0x00 BLUE 0x01 GREEN 0x02 CYAN 0x03 RED 0x04 MAGENTA 0x05 YELLOW 0x06 WHITE 0x07
	REVERSE 0x10 UNDERLN 0x20 BLINK 0x40 HILIGHT 0x80

If the flag show_cur is set, the end of the message line will display the cursor's current row and column, if the message text is short enough to permit it. (See c_vis.)

If a function has been installed (via statfnc) to be called during status line updates, that function is called first. If that function's return is nonzero, the present function assumes that any desired status line display has already taken place, and simply returns.

To clear the message line, use

```
d_msg_line ("", 0);
```

NAME

d_window - display a memory resident window

SYNOPSIS

```
sm_d_window (form_ptr, start_line, start_column);
char *form_ptr;
int start_line;
int start_column;
```

DESCRIPTION

Displays a memory-resident screen at the specified line and column, counting from zero: if 'start_line' is 1, the screen is displayed starting on the second line of the physical display. The argument 'form_ptr' is the address of a character array containing the screen data. You can use the FORM2ASC utility to create a source file containing such an array, which you then compile and link with your application.

This function is similar to r_window, except that the form is already in memory. Thus this routine bears the same relationship to r_window as d_form bears to r_form.

If 'start_line' is negative, this function behaves identically to d_form; 'start_column' is ignored.

RETURNS

0 is returned if no error occurred during display of the form. If the screen is too big for the physical display but no fields need be truncated, this routine displays a warning message and as much of the screen as will fit, and returns 0. -3 is returned if the system ran out of memory, or if the screen contained fields that would not fit within the physical display. The previously displayed screen is restored in this case.

NAME

dtofield - write a double floating point value to the specified field

SYNOPSIS

```
sm_dtofield (field_number, value, format);
int field_number;
double value;
char *format;
```

DESCRIPTION

The double precision variable passed as 'value' is converted to human-readable form, using 'format', and written into the field specified by 'field_number'. If 'format' is equal to 0, the number of decimal places will be taken from a FLOAT or DOUBLE data type edit, if one exists; failing that, from an amount edit, if one exists; or failing that, will default to 2. Data longer than the destination field will be truncated.

The 'format' string should be in the style of printf(); refer to any C language manual for particulars. Here are just a few examples:

"%8g" "%9E" "%4.2f"

RETURNS

-1 is returned if the field name is not found, or if the field, element, or item number is out of range. 0 is returned otherwise.

VARIANTS AND RELATED FUNCTIONS

```
sm_o_dtofield (field_number, occurrence, value, format);
sm_n_dtofield (field_name, value, format);
sm_i_dtofield (field_name, occurrence, value, format);
sm_e_dtofield (field_name, element, value, format);
```

NAME

menu_proc - get menu selection

SYNOPSIS

```
key = sm_menu_proc (type);
int key;
char type;
```

DESCRIPTION

Allows you to tab, backtab, arrow, and scroll through a menu screen, and select an item from it. The entry under the cursor is displayed in reverse video. The routine returns to the calling program when the user hits the TRANSMIT key, the EXIT key, any function key (PF, SPF, or APP), or a sequence of characters that uniquely match a menu entry (see mp_string).

Hitting a key that matches the first character of a menu entry on the screen causes the cursor to be positioned to that entry. If 'type' is UPPER (or LOWER), any alphabetic keyboard entry is translated to upper (or lower) case before a match is attempted; otherwise, no translation is done. The search always starts at the beginning of the menu, and ignores off-screen data; to see off-screen menu items you must use the scrolling keys.

Each menu selection must be defined as initial data in an unprotected, return entry field. Furthermore, unless you change the default setting by calling mp_string, each selection must begin with a unique character.

Two auxiliary functions, mp_options and mp_string, can alter the behavior of the cursor; refer to their definitions.

See the JYACC FORMAKER Author's Guide for a detailed discussion of menu creation and use.

RETURNS

```
key = 0 if the cursor is not in a field.
    = the ASCII value of the first character of the selected menu entry.
    = EXIT (as defined in keys.h) if the exit key was hit.
    = the translated value of any function key hit.
```

VARIANTS AND RELATED FUNCTIONS

```
sm_choice (type);
sm_mp_options (wrap, vert_arrow, horiz_arrow);
sm_mp_string (option);
```

NAME

mp_options - set menu_proc options

SYNOPSIS

```
sm_mp_options (wrap, vertical_arrow, horizontal_arrow);
int wrap;
int vertical_arrow;
int horizontal_arrow;
```

DESCRIPTION

This function takes three parameters. The first determines whether the arrow keys wrap, that is, whether the cursor procedes from the rightmost field around to the leftmost on right arrow (and so forth). The TAB and BACKTAB keys always wrap.

The next two parameters influence which field the arrow keys land you in when wrapping is not imminent. All three parameters must be passed, even if default values are desired.

The mnemonics listed below are defined in "sm_defs.h;" they are the same as those used by ok_options.

wrap:

OK_NOWRAP	No wrapping. The terminal beeps if an attempt is made to arrow past the edge of the current form (or window). OK_NOWRAP is overridden by OK_TAB and equivalent settings.
OK_WRAP	Default. The arrow keys wrap.

vertical_arrow (up and down arrow keys):

OK_NXTLINE	The cursor will be positioned to the closest field whose line is closest to the current line.
OK_FREE	Default. Same as OK_NXTLINE.
OK_RESTRICT	The arrow keys are not operative.
OK_SWATH	The cursor will be positioned to the closest field that overlaps the "swath" containing the current field.
OK_COLM	Same as OK_SWATH.
OK_NXTFLD	The cursor will be positioned to the field closest to the current line and column. The calculation uses the diagonal distance, assuming that the terminal has a 5 to 2 aspect ratio.
OK_TAB	The arrow keys behave like tab and backtab.

horizontal_arrow (left and right arrow keys):

OK_TAB	The arrow keys behave like tab and backtab. OK_FREE Default. Same as OK_TAB.
OK_RESTRICT	The arrow keys are not operative.
OK_COLM	The cursor will be positioned to the closest field on the current line.
OK_SWATH	Same as OK_COLM.
OK_NXTLINE	The cursor will be positioned to the closest of those fields whose column is closest to the current column.
OK_NXTFLD	The cursor will be positioned to the field closest to the current line and column. The

calculation uses the diagonal distance, assuming that the terminal has a 5 to 2 aspect ratio.

RETURNS

0 is returned if the parameters are valid. -1 is returned otherwise. In this case, no options are set.

routine wishes to output a message, it should print to the terminal in the normal way.

The single parameter passed to this routine is a pointer to a character array (currently of size 30). If there is a SMTERM or TERM variable in the environment, the array contains the terminal mnemonic specified there. uinit is free to modify that mnemonic, and the new value will be used in video and keyboard initialization. If uinit indicates an error by returning a nonzero value, or if the key or video initializations fail, JYACC FORMAKER will exit to the operating system.

If there is no SMTERM or TERM variable, the array passed to this routine contains a null string. If uinit returns an error, or the video or key initialization fails, the user will be prompted for a terminal type. User, key and video initialization will then be retried, even if they previously succeeded. Failure on the second attempt causes initcrt to exit to the operating system.

Applications can use uinit to:

- Determine and set the terminal type (particularly on systems with no environment).
- Set screen manager options, such as behavior of the cursor motion keys (ok_options) and various message display attributes.

The library function resetcrt is called to restore the operating system and terminal to their initial states. The macro SYSTEMRESET in "machine.h" performs the operating system reset; the RESET string in the video file performs the terminal reset. A user reset routine, ureset, is then called. A stub routine is supplied in the library, and may be replaced by a user-written routine.

This function is intended to be used to "clean up." It can be used to close communication lines, restore operating system windows, etc. It receives no parameters, and its return value is ignored.

8.3 Memo Text Edits

Memo text edits are not function hooks; rather, they can used by an application programmer to attach arbitrary information to a field. These edits are ignored by the library routines; any testing of them must be done in the application program (possibly in an attached function). An example of memo edit use follows.

Suppose a form contains fields whose contents are interdependent, such as state abbreviation and zip code. The zip code field might have an attached function that performs a validation based on the state abbreviation. However, if the zip code was validated and the operator then changed the state abbreviation, the "validated" zip code could become invalid. A simple solution would be to attach the following function to the state abbreviation field:

- checking for input on an alternate port (such as a digitizer)
- updating a time-of-day display
- monitoring the status of external communication lines

Caution is necessary when using such a routine to display information about arbitrary asynchronous events inside an application, because it is called only when the keyboard is open. Usually this will present no problem, because interactive applications normally close the keyboard only for brief periods, during which there is no opportunity for operator intervention. But the application designer should be aware that transient conditions occurring while the keyboard is closed would not be detected.

8.5 The Status Line Function

The status line function is a routine that JYACC FORMAKER calls whenever the terminal's status line is about to be updated. It is supplied principally in case a particular terminal's status line cannot be accessed with any of the standard video sequences, in which case special programming is necessary to support the library routines that access the status line.

Before calling the status line function, JYACC FORMAKER loads the global buffer sm_line25_buf with the text that is to appear on the status line. If the status line function returns a nonzero value, JYACC FORMAKER assumes that the status line has been updated and skips further processing; otherwise, it proceeds with to display text on the status line as normal.

The application status line function must be installed with a call to the library routine statfnc.

8.6 Video Processing Functions

JYACC FORMAKER channels all screen commands through vproc and vseq. The arguments to the former are a logical screen operation and its parameters (from zero to 11 in number). The operations and their parameters are described in the Video Manual, and their codes can be found in the file "sm_video.h". The vproc supplied in the JYACC FORMAKER library does nothing except call vseq, passing a pointer to its (vproc)'s first argument as the only parameter; it is provided in case application programs want to do special processing on certain video sequences.

An application-supplied vproc can examine the video operation code and do special processing; it can then either return, or pass the address of the code on to vseq for normal output. There is no provision for defining extra codes in the video file; if the application designer wants to create more logical screen operations, they must be defined in application headers and interpreted by the application's vproc. JYACC FORMAKER ignores any value returned from vproc.

Index

In this Index, library functions are displayed in boldface, without the prefixes specific to the language interface. Video and setup file entries appear in ELITE CAPS, while utility programs and JPL commands are in elite lower-case. Function key names are in ROMAN CAPS.

W
window 3-43

skipsomething