FORMS2

Forms for z88dk C for Amstrad PCW

The standard input procedures of C via scanf may be sufficient for simple tasks, but we need more advanced functions for more demanding applications. For example, when entering whole numbers, only the digits 0 to 9 should be allowed with a freely definable maximum input length. Or the input of a file name should be limited to the 8.3 format (plus drive specification).

FORMS2 is a library for the definition and operation of input forms under z88dk C. It consists of the main files forms2.c and forms2.h, along with the helpers box.c, convert.c, and input.c and their corresponding header files, plus the header file mytypes.h. To use the functions provided the client program must include the header file forms2.h:

```
#include "forms2.h"
```

A form is made of a doubly linked list of form elements. A form element is either a field (feField) or a button (feButton). Properties of the form elements include positioning on the screen, the text entered or the activation of a button.

Field – feField

The following field types are supported:

Field type	Description	Sample input
ftText	Text with up to 255 characters	Hello, world!
ftNumNatural	Natural numbers, only the digits 0 - 9 are allowed	2533
ftInteger	Whole numbers, the digits 0 - 9 and a sign (+ or -) are allowed	-1398
ftReal	Real numbers, the digits 0 – 9, a decimal character[1], a sign (+ oder -) and, if necessary, an exponent are allowed	2,997925E+5
ftDate	Date in the format DD.MM.YYYY	18.03.2021
ftFileName	File name in 8.3 format with optional drive specification	M:HELLO.TXT

Fields have a type (see above), a length (fieldLength), a caption, the position of the caption (xCaption, yCaption), the position of the input area (xInput, yInput), the entered text (txt) and a mandatory field property. Caption and text are dynamic variables, their memory requirement is caption or text length + 1 bytes.

Button – feButton

Buttons have a type (ok, cancel, other), a position (x, y), a width, a caption and an activation state (activated). They are activated with the Return key. If a button is left with the Tab key or with one of the arrow keys, it is not activated.

---

1 The decimal character is set via the procedure setDecimalChar.

`ok` and `cancel` buttons are commonly used to confirm or cancel form entry. However, you can also use `other` buttons as needed.

Activating a button ends the form entry. If a button of the type Cancel was pressed, then the check of the mandatory fields is skipped; with the types `ok` and `other`, the form entry is only terminated if all mandatory fields are filled in.

Initialisation - Definition of the form elements

One design goal of the form library was to make the use of the functionality as simple as possible and to hide as much of the internals as possible. The initialisation of the form is therefore done via the function **`initForm`** with specification of the form index; for the definition of the form elements, a function called **`initFormElements`** must be written with the form index as a parameter. In the latter, the elements are defined by calling **`addField`** or **`addButton`** and get attached to each other using **`newAndLinkWithPrevElement`**. Details follow in the sample application `test3` (see below).

Syntax:

```
addField(ftText, 20, 'Name', 10, 6, 17, 6);
          |      |      |     |  |   |  |
          |      |      |     |  |   |  └─ Y input
          |      |      |     |  |   └──── X input
          |      |      |     |  └──────── Y caption
          |      |      |     └─────────── X caption
          |      |      └───────────────── caption
          |      └──────────────────────── field length
          └─────────────────────────────── kind

addButton(Ok, 1, 18, 15, 'OK');
          |   |   |   |    |
          |   |   |   |    └─ caption
          |   |   |   └────── width
          |   |   └────────── Y
          |   └────────────── X
          └────────────────── kind

newAndLinkWithPrevElement
```

Furthermore, the decimal point for the input of real numbers is set with **`setDecimalChar`**, e.g. `setDecimalChar(',')`, and the position of the message line is defined by **`setMessageLine`**, e.g. `setMessageLine(31)`. This is usually done at the beginning of the main program.

Memory management

The forms functionality works with dynamic memory allocation. The allocated memory must be released when a form and its variables are no longer used by calling **`destroy`** with the form index as the only parameter. After the `destroy` call the form data is no longer available.

Start a form

The form is started with the function **`processForm`**. The call is usually made directly after the initialisation via `initForm`.

Form operation

When the form has been started via `processForm`, the labels and controls appear at the specified positions on the screen. The cursor is positioned in the first element and input can be started. Form operation should be largely intuitive.

Input field

An input field begins with the caption. If the mandatory property is set, i.e. if it is a mandatory field, this is indicated by an asterisk (*) in front of the caption. The maximum length of the entry is indicated by a line behind the caption. At the beginning, the cursor is at the start of the field and moves to the right as entries are made. It can be moved within the input area with the left and right arrow keys to insert data elsewhere. The backspace and delete keys can be used to delete entries at the cursor position to the left and right respectively. Pressing the CAN key clears the field completely. Pressing the Tab or Down arrow key moves the cursor to the next field; pressing the Up arrow key moves it to the previous field.

Button

On buttons, the only input option is the return key to activate. The arrow keys or the Tab key are used to leave the button without activating it.
Activating a button ends the form entry, unless at least one mandatory field is not filled in and the button pressed is not a Cancel button. In this case, an error message is displayed in the message line.

Accessing form elements

Two functions make it possible to conveniently access certain form elements: with **`getFormElement`** one gets the nth element in the list, with **`getField`** the nth field (whereby only input fields are counted here) and with **`getButton`** the nth button (whereby only the buttons are counted here).
For example, with `element = getFormElement(5)` you get a pointer to the fifth form element as `FormElement*`. Or with `button = getButton(1)` you get the first button in the form as `Button*`. Afterwards you can access the properties of the elements via the pointers.

Example getFormElement

```
FormElement *elem;
Field *field;
elem  = getFormElement(3); // a text field, i.e. elementType = ftText
field = elem->field;
printf("Field content: %s\n", field->txt);
```

Example getField

```
Field *field;
field = getField(5);
printf("Field content: %s\n", field->txt);
```

Example getButton

```
Button *button;
button = getButton(2);
printf("Button activated: ");
if (button->activated) then
  printf("yes");
else
  printf("no");
```

Further form functions

formCancelled: abort check

If a Cancel button has been pressed, then the form data should not be used. The cancellation can be easily checked with the **`formCancelled`** function following `processForm`:

```
if (formCancelled) {
  // actions upon termination
}
else {
  // actions upon confirmed input *)
}
```

isActivated: check whether a button has been activated

If you want to check whether a particular button has been activated, you can do this very simply with **`isActivated`**:

```
if isActivated(2) {
  // actions with button no. 2 activated
}
```

Data conversion

Form fields always save the entered values as text. For example, if you have entered a value in an integer field, it is still stored as a string in the form field, not as an integer. The conversion is supported by functions in `convert.h`:

Funktion	Beschreibung
stringToInteger	string to integer value; 0 is returned if an error occurs
stringToFloat	string to real value; 0 is returned if an error occurs

setFieldText

Form fields can be preset, which is already possible with the previous functions. But it is particularly easy with **`setFieldText`**, here are a few examples:

```
setFieldText(1, "my preset text"); // text field – ftText
setFieldText(2, "-47"); // number field – ftInteger
setFieldText(5, "31.12.2021"); // date field – ftDate
```

There is no format check during presetting, i.e. the developer must ensure that the field text meets the format requirements of the field type.

Sample application

The sample application creates two forms one after the other and starts them. After finishing the input with the OK or the Cancel button, the field contents and the activation state of the buttons are displayed in each case.

```
#include <stdio.h>
#include <conio.h>
#include "forms2.h"

FormElement *element;
char msg[12];

// Initializes the form elements. This procedure is called from initForm.
void initFormElements(uint8_t formIdx) {

    switch (formIdx) {
        case 1:
            addField(ftText, 20, "Name", 10, 6, 17, 6, false);

            newAndLinkWithPrevElement();
            addField(ftInteger, 5, "Int", 10, 8, 17, 8, true);

            newAndLinkWithPrevElement();
            sprintf(msg, "Size in cm", CH_LC_OUML, CH_SZ);
            addField(ftNumNatural, 3, msg, 10, 10, 24, 10, false);

            newAndLinkWithPrevElement();
            addField(ftReal, 10, "Real", 10, 12, 24, 12, false);

            newAndLinkWithPrevElement();
            addField(ftDate, 10, "Date of birth", 10, 14, 24, 14, false);

            newAndLinkWithPrevElement();
            addField(ftFileName, 14, "Filename", 10, 16, 24, 16, false);

            newAndLinkWithPrevElement();
            addButton(ok, 1, 18, 15, "OK");

            newAndLinkWithPrevElement();
            addButton(cancel, 20, 18, 15, "Cancel");

            break;
         case 2:
            addField(ftDate, 10, "Date", 15, 6, 25, 6, false);

            newAndLinkWithPrevElement();
            addButton(ok, 15, 9, 15, "OK");

            newAndLinkWithPrevElement();
            addButton(cancel, 33, 9, 15, "Cancel");

            break;
    }
}


// Writes the current field's text and moves to the next element.
void nextField() {
  printf("%s\n", element->field->txt);
  element = element->next;
}
```

```
// Writes the current button's activation status and moves to the next element.
void nextButton() {
  if (element->button->activated)
      puts("yes");
  else
      puts("no");
  element = element->next;
}


// Displays a headline.
void headline() {
  clearScreen();
  puts("FORMS2 test\n\n\n");
}


int main(void) {
  setDecimalChar(',');
  setMessageLine(31);

  headline();
  puts("*** Form 1 ***");

  initForm(1);
  setFieldText(1, "Preset text");

//  printFormElements(); // DEBUG

  processForm();

  gotoXY(0, 22);
  element = getFormFirstElement();
  printf("Name         : "); nextField();
  printf("Int          : "); nextField();
  printf("Size         : ", CH_LC_OUML, CH_SZ); nextField();
  printf("Real         : "); nextField();
  printf("Date of birth: "); nextField();
  printf("Dateiname    : "); nextField();
  printf("OK           : "); nextButton();
  printf("Cancel       : "); nextButton();

  if (formCancelled()) {
    printf("Cancelled\n");
  }
  else {
    printf("Confirmed\n");
  }

  destroyForm(1);
  waitForKey();

  headline();
  printf("*** Form 2 ***");

  initForm(2);
  processForm();

  gotoXY(0, 22);
  element = getFormFirstElement();
  printf("Date  : "); nextField();
  printf("OK    : "); nextButton();
  printf("Cancel: "); nextButton();
```

```
  if (formCancelled) {
    printf("Cancelled\n");
  }
  else {
    printf("Confirmed");
  }

  destroyForm(2);

  waitForKey();

  return 0;
}
```