NAME

form_field_validation - data type validation for fields

SYNOPSIS

#include <form.h>

void *field_arg(const FIELD ***field);**
FIELDTYPE *field_type(const FIELD ***field);**
int set_field_type(FIELD ***field, FIELDTYPE ***type**, ...);**

/* predefined field types */
FIELDTYPE *TYPE_ALNUM;
FIELDTYPE *TYPE_ALPHA;
FIELDTYPE *TYPE_ENUM;
FIELDTYPE *TYPE_INTEGER;
FIELDTYPE *TYPE_NUMERIC;
FIELDTYPE *TYPE_REGEXP;
FIELDTYPE *TYPE_IPV4;

DESCRIPTION

By default, no validation is done on form fields. You can associate a form with with a *field type*, making the form library validate input.

field_arg

Returns a pointer to the field's argument block. The *argument block* is an opaque structure containing a copy of the arguments provided in a **set_field_type** call.

field_type

Returns a pointer to the *field type* associated with the form field, i.e., by calling **set_field_type**.

set_field_type

The function **set_field_type** associates a field type with a given form field. This is the type checked by validation functions. Most field types are configurable, via arguments which the caller provides when calling **set_field_type**.

Several field types are predefined by the form library.

Predefined types

It is possible to set up new programmer-defined field types. Field types are implemented via the **FIELDTYPE** data structure, which contains several pointers to functions.

See the **form_fieldtype**(3X) manual page, which describes functions which can be used to construct a field-type dynamically.

The predefined types are as follows:

TYPE_ALNUM
Alphanumeric data. Required parameter:

- a third **int** argument, a minimum field width.

TYPE_ALPHA
Character data. Required parameter:

- a third **int** argument, a minimum field width.

TYPE_ENUM
Accept one of a specified set of strings. Required parameters:

- a third **(char **)** argument pointing to a string list;
- a fourth **int** flag argument to enable case-sensitivity;
- a fifth **int** flag argument specifying whether a partial match must be a unique one. If this flag is off, a prefix matches the first of any set of more than one list elements with that prefix.

The library copies the string list, so you may use a list that lives in automatic variables on the stack.

TYPE_INTEGER
Integer data, parsable to an integer by **atoi**(3). Required parameters:

- a third **int** argument controlling the precision,
- a fourth **long** argument constraining minimum value,
- a fifth **long** constraining maximum value. If the maximum value is less than or equal to the minimum value, the range is simply ignored.

On return, the field buffer is formatted according to the **printf** format specification ".*ld", where the "*" is replaced by the precision argument.

For details of the precision handling see **printf**(3).

TYPE_NUMERIC

Numeric data (may have a decimal-point part). Required parameters:

- a third **int** argument controlling the precision,
- a fourth **double** argument constraining minimum value,
- and a fifth **double** constraining maximum value. If your system supports locales, the decimal point character must be the one specified by your locale. If the maximum value is less than or equal to the minimum value, the range is simply ignored.

On return, the field buffer is formatted according to the **printf** format specification ".*f", where the "*" is replaced by the precision argument.

For details of the precision handling see **printf**(3).

TYPE_REGEXP

Regular expression data. Required parameter:

- a third argument, a regular expression **(char *)** string. The data is valid if the regular expression matches it.

Regular expressions are in the format of **regcomp** and **regexec**.

The regular expression must match the whole field. If you have for example, an eight character wide field, a regular expression "^[0-9]*$" always means that you have to fill all eight positions with digits. If you want to allow fewer digits, you may use for example "^[0-9]* *$" which is good for trailing spaces (up to an empty field), or "^ *[0-9]* *$" which is good for leading and trailing spaces around the digits.

TYPE_IPV4

An Internet Protocol Version 4 address. Required parameter:

- none

The form library checks whether or not the buffer has the form *a.b.c.d*, where *a*, *b*, *c*, and *d* are numbers in the range 0 to 255. Trailing blanks in the buffer are ignored. The address itself is not validated.

This is an ncurses extension; this field type may not be available in other curses implementations.

RETURN VALUE

The functions **field_type** and **field_arg** return **NULL** on error. The function **set_field_type** returns one of the following:

E_OK
The routine succeeded.

E_SYSTEM_ERROR
System error occurred (see **errno**(3)).

SEE ALSO

curses(3X), **form**(3X), **form_fieldtype**(3X), **form_variables**(3X).

NOTES

The header file **<form.h>** automatically includes the header file **<curses.h>**.

PORTABILITY

These routines emulate the System V forms library. They were not supported on Version 7 or BSD versions.

AUTHORS

Juergen Pfeifer. Manual pages and adaptation for new curses by Eric S. Raymond.