NAME

ccfe form − the Curses Command Front-end form definition file.

DESCRIPTION

The Curses Command Front-end form is a tool which permit to assist the user to valorize options and arguments requested for the execution of a O.S. command or script, which can be into an external file or embedded in the form. It is based on the form library, the curses extensions for programming forms: please see the **ncurses**(3X) and **form**(3X) manpages. A *ccfe menu* is basically an ordered collection of fields with an *action* to execute when the form is posted. Each field can be associated to an option or an argument.

SYNOPSIS

A form is definited in a file with name terminated with the *.form* extension, and the following attributes can be used in that file:

```
title {}
top {}
path {}
init {}
field {}
field {}
 ...
field {}
action {}
```

(some of them are optional).
Comment lines begin with the "**#**" character. Empty lines are ignored. Comments embedded in lines are not permitted so you can use the "**#**" character in commands and text of the form. All keywords are case insensitive.

GENERAL FORM ATTRIBUTES

The following attributes concern the static parts of the form.

title {*STRING***}**

Title of the form to display.
It is also used as *fast path description* when **ccfe(1)** is called with **-f** option.
If title attribute is omitted, it will be used the *description* of the item selected from the calling menu, but if omitted in a form called with a *fast path* (see ccfe(1)), then no title will be used.
Optional: Yes
Examples:

```
title { Change user password expiry information }
```
or
```
title {
  Change user password expiry information
}
```

top { *string* [\n*string*] **}**

Text lines (max 2) to display on the top of the form, between the title and the fields. If omitted, the lines of the parameter **form_top_msg** in **ccfe.conf**(5) will be displayed.
Optional: Yes
Example:

```
top {
  Type or select a value for the entry field.
  Press Enter AFTER making all desired changes.
}
```

path {*list***:***of***:***pathnames***}**

List of extra path (added to environment variable PATH) to search binaries calling the O.S. in the following contexts:

- field.default
- field.list_cmd
- form.init
- form.action

Note: path{} is used only to search binaries, and not for forms and menus.
Optional: Yes
Formato:
- LIST:OF:PATHNAMES has the character ':' as separator;
- LIST:OF:PATHNAMES can contains relative and absolute subdirectories;
- le directory con path relativo sono relativizzate a $SCREEN_DIR

Example:

```
path { ldap_lib }
```

FIELD DEFINITION

The **field {}** form attribute defines a field of the form. It must appear at least one time in a form definition file. Every field definition is a collection of attributes, with the following syntax:

```
field {
        attribute    = value
        attribute    = value
                      ...
        attribute    = value
}
```

Valid field attributes are explained later with the following format:

attribute_name

Attribute description.

Values: Notes on the syntax of the value of the field. If there is a list of values, the first element of the list is the default. The **""** value represents the *empty string* (or *null string*) value.
Required: Yes or *no*, depending if the attribute is mandatory or not.
Types: List of field **type** for which this attribute is valid.

id

Identifier of the field used to refer its value by **init{}** and **action{}** attributes. It must be unique in the scope of the form and cannot be one of the reserved values **CCFE_REMOVE_FIELDS**,**CCFE_ENABLE_FIELDS**,**CCFE_DISABLE_FIELDS**,**ARGV**x (please see **FORM INITIALIZATION** and the **form action type** in **FORM ACTION** below). It is also used as reference for contextual help (please see the **ccfe_help**(5) manpage).

Values: A string composed by characters **A-Z**,**a-z**,**0-9**,**-**,**_**. It is case sensitive.
Required: Yes
Types: All

label

Prompt string of the field.

Values: A string of any charachter.
Required: Yes
Types: All

enabled

Flag to enable or disable the field. It is not possible to move the cursor to a disabled field, and then to insert a value. Disabled fields ca be used to insert a comment or a simple label in the form.

Values: **YES**|**NO**
Required: No
Types: All

len

Size of the field (in characters). It is ignored for the fields of type **BOOLEAN** and **NULLBOOLEAN**, because it is automatically XX_set up_XX. If "hscroll=YES", then the specified length is a "window" to scroll on the value of the field, which can be more longest of that size.
Required: No
Types: **NUMERIC**, **STRING**, **UCSTRING**
Default: 20

hscroll

Please see the previous attribute.

Values: **NO**|**YES**
Required: No
Types: **NUMERIC**, **STRING**, **UCSTRING**

type

Set of the characters accepted in the field. If none of the following is specified, the **STRING** is used. Available field types are:

NUMERIC
""|0-9,"+","-",",","."
Note that if you want a default value of zero instead the *empty string* value, you can use the "**default = const:0**" field attribute or the **init{}** form attribute (please see **FIELD VALUE INITIALIZATION** below).

BOOLEAN
NO|**YES**

NULLBOOLEAN
NO|**YES**|**""**
Like the previous type, but can also assume the "null" (or "nothing") value, represented by the *empty string* value.

STRING
Alphanumeric string. Default value: *empty string*.

UCSTRING
Alphanumeric string converted in upper case. Default value: *empty string*.

The fields of type **BOOLEAN** and **NULLBOOLEAN** have the following additional features:

- Navigation between all admitted values with the <Tab> key;
- Automatic setup of the **list_cmd** attribute.

Required: Yes

option

Every shell command accepts options and arguments: this attribute let you generate a command option depending on the value inserted in the field. Every occurrence **%{** *field_id***}** in the **action{}** attribute will be substituted by *OPTION VALUE*. If the field value contains blanks between tokens, then *OPTION VALUE*

will be repeated for every token, unless is specified to quote it. To quote the value of a field you must end the option with a double quote character (") or a single quote character ('), as explained in the examples below.
Additional features:

- If field type is **BOOLEAN**, instead a single value, you may list the two values to return if the user select **YES** or **NO**, and then every occurrence **%{** *field_id* **}** in **action{}** will be substituted by it.
  For example, the definition:

  ```
  field {
    id     = LOCK
    label  = Lock "guest" user account?
    type   = BOOLEAN
    option = -L,-U
  }
  action {
    run:usermod %{LOCK} guest
  }
  ```

  will execute the command:

  - "**usermod -L guest**" if the field value is **YES**
  - "**usermod -U guest**" if the field value is **NO**.

  The second value of the **option** attribute is optional: it is not required for options which must appear only if the user select for them the value **YES**; for example, the definition:

  ```
  field {
    id     = LONG
    label  = Use a long listing format
    type   = BOOLEAN
    option = -l
  }
  action {
    run:ls %{LONG}
  }
  ```

  will execute the command:

  - "**ls -l**" if the field value is **YES**
  - "**ls**" if the field value is **NO**.

- If field type is **NULLBOOLEAN**, then every occurrence **%{** *field_id* **}** in **action{}** will be substituted by:

  - "*option* **y**" if the value inserted is **YES**
  - "*option* **n**" if the value inserted is **NO**
  - `""` otherwise.

  For example, the definition:

  ```
  field {
    id    = VG_NAME
    label = Volume Group Name
    type  = STRING
    len   = 32
  }
  field {
  ```

```
  id      = AVAILABLE
  label   = Activate Volume Group?
  type    = NULLBOOLEAN
  default = const:      # see the default attribute
  option  = -a
}
action {
  run:vgchange %{AVAILABLE} %{VG_NAME}
}
```

will execute the command:

- "**vgchange -a y vg00**" if the *AVAILABLE* value is **YES**.
- "**vgchange -a n vg00**" if the *AVAILABLE* value is **NO**.
- "**vgchange vg00**" if the *AVAILABLE* value is `""`.

- If field type is **STRING** or **UCSTRING** and multiple words are inserted in the field value, then every occurrence **%{** *field_id***}** in **action{}** will be substituted as the following examples:

```
field {
  id     = TEST_OPT
  label  = Value(s)
  type   = STRING
  option = see the following table
}
action {
  run:mycmd %{TEST_OPT}
}
```

box, center, tab (@); c | c | c C | L | L. **option**@Value in field@Command executed = -a@one@mycmd -a one -a@one two@mycmd -a one -a two -b"@one@mycmd -b "one" -b"@one two@mycmd -b "one two" -b’@one@mycmd -b ’one’ -b’@one two three@mycmd -b ’one two three’ --option-c@one@mycmd --option-c one --option-c@one two@mycmd --option-c one --option-c two --option-d="@one@mycmd --option-d="one" --option-d="@one two@mycmd --option-d="one two" --option-d=’@one@mycmd --option-d=’one’ --option-d=’@one two@mycmd --option-d=’one two’ --option-e=@one@mycmd --option-e=one --option-e=@one two@mycmd --option-e=one --option-e=two

Obviously, multiple fields with **option** attribute can appear in a form definition, mixed with fields without it, so the user can enter options and arguments requested by the command defined in the **action** of the form. Please note that a *blank* character is insterted before every occurrence, so it is advisable to not insert blanks between option referenced by token **%{** *field_id***}** in the **action{}** of the form. For example, it is preferable the syntax

```
action {
  wget %{OPT_OUTPUT_FILE}%{OPT_DEBUG} %{URL_LIST}
}
```

instead of

```
action {
  wget %{OPT_OUTPUT_FILE} %{OPT_DEBUG} %{URL_LIST}
}
```

Required: No
Types: All

hidden

Hide the value of the field by substituting its characters with asteriscs. Useful for fields used to accept passwords.

Values: **NO|YES**
Required: No
Types: **STRING**, **UCSTRING**

required

Mark the prompt with '*' to indicate which is mandatory to insert a value to execute the form action. It is used only for informational purposes: but no checks are done to the value of the field (this job is demanded to the **action**), but the action of the form will not executed until every required field is valorized.

Values: **NO|YES**
Required: No
Types: All

ignore_unchgd

If the value of the field was NOT MODIFIED by the user, then the substitution of every occurrence of **%{** *field_id* **}** in the action will not be done. This is useful when you want to run commands specifying only the arguments or options related to fields effectively entered or changed by user. For example, you can post a form with default values but you may want to execute the command in the action with the changed fields only.

Values: **NO|YES**
Required: No
Types: All

htab

Right indents the label of the field. This integer value defines how many **tabs** are used for indentation. A *tab* is 2 columns, so, for example:

```
htab = 2
```

right shifts 4 columns the label of the field. It can be used to group logically some fields under another field or under a text separator (please see **separator{}** attribute below).

Values: <positive integer value>
Required: No
Types: All

vtab

Insert blank lines before the field e the preceeding. In a multi-page form, if the field bottom of the page has this attribute, then it will be the first of the next page, but without being preceded by blank lines.
Example:

```
vtab = 1    # skip 1 line from the previous field
```

Values: <positive integer value>
Required: No
Types: All

default

Value assigned to the field when the form is posted. It overrides the default value dependent from the field **type**, and is overridden by the **init** form attribute (please see **FORM INITIALIZATION** below).

Values: must follow the syntax

*source***:***value*

where

source can be one of the keywords:

command
The *value* is the stdout produced by an arbitrary command.

const The *value* is an arbitrary constant. Valid values depend from the **type** of the field. All characters after **:** are used (alphanumeric, punctuation, etc). For this reason strings must not be enclosed in single or double quotes.

value Constant or command, depending from *source*. An invalid value (for example a string in **BOOLEAN** field or a command which return an error) will fill the field with the string "**ERROR!**".

Examples:

```
default = command:date "+%Y-%m-%d"
default = const:192.168.1.4/24
default = const:staff adm lpadmin
default = command:id -Gn | tr " " "\n" | grep -v "‘id -gn‘" | tr "\n" " "  # Se
default = const:1
default = const:          # "null" value in a NULLBOOLEAN field
default = const:""        # NOT valid in a NULLBOOLEAN field!
```

Required: No
Types: All

list_cmd
Instructions for generate the pop-up list of values when the user press **<list key>**, so he/she can quickly enter in the field an admitted value. Every entry in the list is a blank separated pair of values (160 characters max). The syntax for this attribue is the following:

source-type **:** *list-type* **:** *list-type-arguments*

Where:

source-type can be one of the following:

const *list-type-arguments* is a list of constant values.

command
list-type-arguments is the stdout produced by a shell command.

list-type can be one of the following:

single-val
The user can select only one value from the list.

multi-val
The user can select one or more values from the list. They are returned separated by a *blank* or by the character specified with the **list_sep** attribute, please see its description below.

list-type-arguments can be one of the following:

if *source-type* is **command**, then the list is generated by the
stdout of the O.S. command (it is not admitted a shell script as in the **action {}** or **init {}** attributes), which must be with the following format:

Val1 Descr1
Val2 Descr2
Val3 Descr3

...
ValN DescrN

It is possible to pass the actual value of the form fields as parameters of the command with the syntax **%{***field_id***}** , but only for the fields which type is neither **BOOLEAN** nor **NULL-BOOLEAN**. Please see the examples below.

if *source-type* is **const**, then *list-values* is a list with the format:

"*Val1 Descr1*"**,**"*Val2 Descr2*"**,**"*Val3 Descr3*"**,...**
PP *ValN* is the value which will be put in the field if the item is selected from the list.

DescrN is optional and can contain blanks. It is possible to insert blanks in *ValN* by protecting them with the backslash \, for example:

```
"First\ value Descr1","My\ second\ value Descr2","Val3 Descr3"
```

It is also possible to insert the preferred separators between *ValN* and *DescrN* with a simple trick as in the following examples:

```
"First : 1st description","Second : 2nd description",...
"First --->1st description","Second --->2nd description",...
"First ---> 1st description","Second ---> 2nd description",...
"First\ value : 1st description","Second\ value : 2nd description",...
```

Warning: special characters as <TAB> in *ValN* or *DescrN* strings can confuse the output of the the pop-up menu.

Examples:

1. Simple list of values without description: the definition
   `list_cmd = const:single-val:"root_lv","usr_lv","var_lv","home_lv"`
   will produce the list:

```
root_lv
usr_lv
home_lv
```

2. List of values with description: the definition

```
list_cmd = const:single-val:"90 Expire password after 3 months","180 Expire password after 6 months","-1 Disable user password expiration"
```

will produce the list:

```
90  Expire password after 3 months
180 Expire password after 6 months
-1  Disable user password expiration
```

Selecting the first item the field will be filled with the value *90*, selecting the second item with *180*, and *-1* selecting the last item.

3. List of values with description produced by a O.S. command: the definition

```
list_cmd = command:single-val:cut -d : -f 1,5 --output-delimiter=" " /etc/passwd | sort
```

will produce a list like this (username + gecos fields of `/etc/passwd` file):

```
bin        bin
daemon     daemon
dhcp       DHCP subsystem
```

```
ftp        FTP subsystem
games      games
gdm        Gnome Display Manager
haldaemon  Hardware abstraction layer,,,
hplip      HPLIP system user,,,
irc        Internet Relay Chat subsystem
list       Mailing List Manager
lp         Printer spooler subsystem
mail       Mail subsystem
man        Online manual subsystem
news       NNTP subsystem
nobody     nobody
proxy      proxy subsystem
pulse      PulseAudio daemon,,,
root       Super-user
sshd
sys        sys
uucp       Unix-to-Unix Copy Program subsystem
```

list_sep

This attribute is only used by the **list_cmd** attribute of type **multi-val**: it defines the separator to use in action.

Values: *blank*|**,**|**;**|**:**
Required: No
Types: All

persist

If enabled, the next time the form is loaded, the field will be initialized with the value it has at the last action execution time. Values provided by the **init{}** block and the **default** field attribute are overridden by persistent field value, but with one exception: the first time a form is used, because persistent fields have no value.

Values: **NO**|**YES**
Required: No
Types: All

Example of field definition:

```
field {
  id       = STREE
  label    = LDAP Subtree
  enabled  = NO
  len      = 8
  hscroll  = YES
  type     = STRING
  option   = -s
  hidden   = NO
  required = YES
  htab     = 1
  default  = const:Mail
  list_cmd = command:single-val:userlist.sh %{STREE}
}
```

FIELDS SEPARATOR DEFINITION

The **separator {}** form attribute defines a separator between the fields of the form. Valid separator attributes are:

id

Identifier of the separator used to refer its value by **init{}** and **action{}** parameters. It must be unique in the scope of the form.
If not specified, the reserved value **CCFEFSEP***nnn* will be used (001 <= nnn <= 999).

Values: An upper case string (but in reality it is case insensitive) composed by characters A-Z,0-9,-,_
Required: No

type

Type of separator. There is not a *default separator type* if not specified, so it must be one of the following values:

TEXT A string of characters with left alignment. Like a field value, it is possible to assign a value to a *TEXT separator* with the init{} form attribute. Please see **FIELD VALUE INITIALIZATION** below.

TEXT_CENTER
A string of characters with centered alignment.

LINE A simple line of '-' with screen width.

LINE_DOUBLE
A simple line of '=' with screen width.

The types **TEXT** and **TEXT_CENTER** can be used to put in the form arbitrary text or special instructions for the user.

text

String to display when the separator is of **type = TEXT** or **type = TEXT_CENTER**.
Example:

```
separator {
  type = TEXT
  text = This is a comment between fields of a form!
}
```

Values: A string of any charachter.
Required: No

htab

Right indents the separator. This integer value defines how many **tabs** are used for indentation. A **tab** is 2 columns, so, for example:

```
htab = 2
```

shifts separator 4 columns to right.
It can be used to group logically some fields under a text separator.

Values: <integer value>
Required: No

vtab

Insert blank lines before the separator and the preceeding field (or separator). In a multi-page form, if the separator bottom of the page has this attribute, then it will be the first of the next page, but without be preceded by blank lines.

Example:

```
vtab = 1    # skip 1 line from the previous field/separator
```

Values: <positive integer value>
Required: No

Example of separator definition:

```
separator {
  type = TEXT
  text = HTTP options:
  vtab = 1
}
```

FORM INITIALIZATION

The **init {}** form attribute permits to execute arbitrary commands before the form is posted. It can be a complete shell script or a simple call to an O.S. command. The standard output is used to initialize the value of the fields of the form, as described below. The syntax for this attribue is the following:

init { *init-type***:** *init-arguments* **}**

Where:

init-type
defines how to perform the initializations. Actually, the only supported type is **command**.

init-arguments
is the O.S. command or the script executed with the shell specified by the parameter **shell** in the **ccfe.conf**(5) file.

Using arguments

If the form is called with arguments, they can be referred in the **init {}** block as **%{ARGV*n*}**, exactly as in the **action {}** block; for example, `caller.form` can call a form with

```
action {
  form:called %{HOSTNAME} %{USERNAME}
}
```

and these arguments can be referred by with `called.form` with

```
init { command:
  echo hostname: %{ARGV1}
  echo username: %{ARGV2}
}
```

Fields value initialization

Every line of the standard output produced by *init-arguments* in the format

field_id=*value*

is used to initialize with *value* the field with ID *field_id*, by overriding, if specified, the corresponding value of the **default** field attribute.

Diagnostics and form-post control

If the **init {}** block produces some standard error, it will be displayed in a pop-up window. If the **init {}** block produces an exit status greater than zero, then the form will not be posted. so you can decide to BLAH, BLAH...

```
  DIAGNOSTICA:
    Se i comandi in init{}
    - producono stderr, questo verra’ visualizzato in una finestra pop-up.
    - producono un exit status > 0, allora non verra’ presentata la form
      appena chiamata.
    Questo consente quindi di impedire all’utente di proseguire nel caso
    init{} sia eseguito con errori gravi, oppure di ignorarli.
    La diagnostica puo’ essere disabilitata semplicemente redirezionando
    su /dev/null l’stderr dei comandi in init{}.
Opzionale: Si
Osservazioni: Per come vengono eseguiti i comandi specificati, l’eventuale
              presenza di un apice singolo va specificata con la sintassi ’´’
              Ad esempio, il comando bash:
                echo "Utente gia’ definito nel sistema" > /dev/stderr
              diventa
                echo "Utente gia’´’ definito nel sistema" > /dev/stderr

              Preferire l’uso del costrutto $() anziche’ ‘‘ per l’esecuzione
              di comandi in subshell.
FORM DINAMICHE:
  E’ possibile rimuovere, disabilitare ed abilitare arbitrariamente campi dalla form
  assegnandone gli ID separati da virgole rispettivamente alle seguenti variabili
  fittizie:
    CCFE_REMOVE_FIELDS
    CCFE_ENABLE_FIELDS
    CCFE_DISABLE_FIELDS
  Ad esempio
    init { command:
      echo CCFE_REMOVE_FIELDS=F1,F3
    }
  rimuove dalla form i campi con ID F1 e F3

Esempio:
  init { command:
    if grep -qw $1 /etc/passwd; then
      echo "Utente gia’´’ definito nel sistema" > /dev/stderr
      exit 1     # Impedisce chiamata form successiva
    else
      echo UNAME=$1
    fi
  }
```

Please see also the **default** field attribute.

FORM ACTION

The form attribute **action {}** executes a command with the values typed in the fields of the form. There is no check of this attribute definition, but a form without **action{}** is very useless! The syntax for a *normal* *action* of a form is the following:

action { *action-type*[**(***options***)**]**:** *action-args* **}**

It is also possible to change the action depending the item selected in the menu which called the form: in this case the syntax is the following:

```
action {
        select-item {
                item_id0    : action-type[(options)]: action-args
                item_id1    : action-type[(options)]: action-args
                [...]
                item_idN    : action-type[(options)]: action-args
        }
}
```

NB: con l'item selector posso specificare solamente action
semplici su una sola linea: questo costrutto va utilizzato
per selezionare la form successiva da una form comune
(vedi esempio).

Where:

action-type can be one of the following:

run Execute *action-args* and then show the results in the **Output Browser screen** (please see the **ccfe**(1) manpage). *action-args* can be a simple command or a complete script which will be passed to a command shell for parsing (with "sh -c"). The shell used is the one specified in the **shell** parameter of the `ccfe.conf` configuration file.

form Load and post the form specified in *action-args*. If arguments are present, they can be referred in the **init{}** and **action{}** attributes of the called form using the syntax **%{ARGV***n***}** , with 0 < *n* < 100. All the **%{ARGV***n***}** not passed as arguments have a null string value. Before the name of form, you can specify a path relative to /usr/share/ccfe directory (for example `users.d/ask_user`). A form can call another form which can call another form and so on: it is useful when you want to ask for parameters with special treatment before the final action execution.

system Temporarily exit from **ccfe**, executes *action-args* and returns in **ccfe**. This is useful for actions which need user interaction: for example the **passwd**(1) command. If **wait_key** is specified in *options*, then it is displayed a message reporting the exit status and wait a user keystroke before to return in **ccfe**.

exec Terminate **ccfe** and executes *action-args*. This is useful to return to Operating System after calling external commands or applications, using **ccfe** as application launcher. Be aware that *action-args* are executed directly instead of via a command shell, so invocation errors may not be reported.

options is a comma-separated list of one or more of the following options:

confirm
Open a pop-up window to request a confirm before execute action. The option highlighted by default is "No" (not confirm).

log Save in the logfile the output of the execution of the action. Not valid for *action-type*=**system**.

wait_key
Valid only for *action-type*=**system**: when action is completed, wait for continuing until the user presses a key.

item_idX are item **id**'s defined in the calling menu.

The * metacharacter

The metacharacter ***** expands all fields IDs of the form. This can be useful for fast form deployment: in forms with many fields and a very simple action (f.e. a single command call), you can use the metacharacter "*****" as *field_id*, instead to list one by one all form fields IDs. For example, in a form with three fields with ID *ID1*, *ID2* and *ID3*, the action

```
command %{*}
```

is expanded as

```
command %{ID1}%{ID2}%{ID3}
```

Note that using metacharachter *****:

- Field IDs are expanded in the order of their definition in form;
- *Dynamic fields* are supported;
- In fields with **option** attribute defined, blanks are automatically inserted as described above in **option** attribute explanation;
- In fields without **option** attribute, a blank is automatically inserted before its value.

Notes

To refer to the value of the fields, you must use the syntax **%{** *field_id***}** which will be substituted with the value of the field with ID *field_id* before the execution of the action. *field_id* are case senstitive.

This solution was preferred instead to use environment variables (like $CCFE_FIELD_ID) to pass values to action so to not change the child process environment.

With **select-item {}** it is possible to run an action depending of the ID of the item selected in the menu which called the form.

Examples:

1. Action with an embedded Bourne shell script:

```
action { run:
  if [ "x%{VPN_NAME}" == 'xTECH' ]; then
    conf='tech.conf'
  elif [ "x%{VPN_NAME}" == 'xGUESTS' ]; then
```

```
          conf='guests.conf'
        fi
        sudo /usr/sbin/vpnc $conf
      }
```

2. Ask the user to confirm its request to enable the wireless network:

```
action { run(confirm):sudo /usr/local/scripts/iw-up 2>&1 }
```

3. Call the form `/usr/share/ccfe/ldap.d/chage.form` passing it the value of the fields with attribute **id** SUBTREE and USERNAME, which are processed as **%{ARGV1}** and **%{ARGV2}** by the **init{}** attribute of the form `chage.form` to assign the initial value to the fields with the same ID:

```
action { form:ldap.d/chage %{SUBTREE} %{USERNAME} }
```

Please see the section **FORM INITIALIZATION** above.

4. Temporarily escape from **ccfe** and execute the command **passwd**(1) to change the password of the user specified in the field with **id** USERNAME. Prompt a message and wait the user hit a key before to re-enter in **ccfe**, so the user can see if **passwd**(1) command changed or not the password:

```
action { system:/usr/bin/passwd(wait_key) %{USERNAME} }
```

If **ccfe** is not running by root user, then he can change only its own password.

5. Help users to connect to remote host by selecting it from a list:

```
  field {
      id       = RH
      label    = Remote host
      len      = 35
      type     = STRING
      list_cmd   =   const:single-val:"host1.somewere.com","host2.some-
lan.net"
    }
    field {
      id      = PORT
      label   = Remote port
      len     = 5
      type    = NUMERIC
      default = const:22
    }
    action { exec: telnet %{RH} %{PORT} }
```

6. ask_user.form:

```
field {
        id          = USERNAME
        label       = User name
        type        = STRING
        len         = 8
}
action {
        select-item {
                USERADD
                        : form:users.d/useradd %{USERNAME}
```

```
                    USERMOD
                                 : form:users.d/usermod %{USERNAME}
                    USERDEL   : form:users.d/userdel %{USERNAME}
                    PASSWD     : system(wait_key):passwd %{USERNAME}
            }
    }
```

The definition of menu which called the ask_user.form is the following:

```
item {
        id        = USERADD
        descr     = Create a new user account
        action    = form:users.d/ask_user
}
item {
        id        = USERMOD
        descr     = Modify a user account
        action    = form:users.d/ask_user
}
item {
        id        = USERDEL
        descr     = Remove a user account and related files
        action    = form:users.d/ask_user
}
item {
        id        = PASSWD
        descr     = Change a user's password
        action    = form:users.d/ask_user
}
```

FORM ARGUMENTS

TODO!

DYNAMIC FORMS

It is possible to create forms that change runtime: for example, when the result of the initialization script is a particular value, or when you want to reuse the same complex form for different purposes (for add and edit similar values, for example). This is possible by:

- using the special values **CCFE_REMOVE_FIELDS**,**CCFE_ENABLE_FIELDS**,**CCFE_DISABLE_FIELDS** in **init{}** attribute (please see **FORM INITIALIZATION** paragraph);

- using the syntax **select-item** in **action{}** attribute (please see the **FORM ACTION** paragraph);

- calling other forms with arguments and referring them in **init{}** and **action{}** attributes with **%{ARGV1}**,**%{ARGV2}**,...,**%{ARGVn}** syntax.

SEE ALSO

ccfe(1), **ccfe.conf**(5), **ccfe_menu**(5), **ccfe_help**(5), **curses**(3X), **form**(3X)

BUGS

Many cosmetics, but the tool is stable and usable.