

# Package PrettyCurses v1.0

Xavier Caruso

août 2002

## Table des matières

<b>Introduction</b>	<b>2</b>
<b>PrettyCurses</b>	<b>3</b>
<b>PrettyCurses::default_french</b>	<b>7</b>
<b>PrettyCurses::utils</b>	<b>8</b>
<b>PrettyCurses::Message</b>	<b>10</b>
<b>PrettyCurses::Corrections</b>	<b>11</b>
<b>PrettyCurses::Widgets</b>	<b>13</b>
<b>PrettyCurses::Caption</b>	<b>17</b>
<b>PrettyCurses::TextField</b>	<b>18</b>
<b>PrettyCurses::TextMemo</b>	<b>21</b>
<b>PrettyCurses::MultiFields</b>	<b>24</b>
<b>PrettyCurses::Menu</b>	<b>27</b>
<b>PrettyCurses::directory</b>	<b>32</b>
<b>PrettyCurses::CheckBox</b>	<b>35</b>
<b>PrettyCurses::Button</b>	<b>37</b>
<b>PrettyCurses::Form</b>	<b>38</b>

# Introduction

## Description

`PrettyCurses v1.0` est une librairie, fonctionnant sous `perl`, version `5.x`. Elle est une extension de la librairie `Curses` et implémente un certain nombre de widgets utilisés fréquemment.

## Licence

Ce programme est un logiciel libre ; vous pouvez le redistribuer et/ou le modifier conformément aux dispositions de la Licence Publique Générale GNU, telle que publiée par la Free Software Foundation ; version 2 de la licence, ou encore (à votre choix) toute version ultérieure.

Ce programme est distribué dans l'espoir qu'il sera utile, mais *sans aucune garantie* ; sans même la garantie implicite de *commercialisation* ou d'*adaptation à un objet particulier*. Pour plus de détail, voir la Licence Publique Générale GNU.

Vous devez avoir reçu un exemplaire de la Licence Publique Générale GNU en même temps que ce programme ; si ce n'est pas le cas, écrivez à la Free Software Foundation Inc., 675 Mass Ave, Cambridge, MA 02139, Etats-Unis.

## Installation

Rien de particulier n'est prévu pour l'installation. Il suffit en fait de copier, en conservant l'arborescence, tous les fichiers `*.pm` distribués dans un répertoire que l'on aura pris soin de rajouter dans la variable globale `@INC` s'il n'y était pas déjà.

Les pages d'aide regroupés dans ce document sont incluses dans les fichiers source sous le format `pod`. La commande `pod2man(1)` permet de créer des pages directement lisibles par la commande `man(1)`.

## Auteur

Je m'appelle Xavier Caruso. Vous pouvez m'écrire à l'adresse `<xavier.caruso@ens.fr>`.

## Avertissement

Cette librairie a été écrite assez rapidement et n'a pas trop été testée. Il est donc fort probable qu'un nombre encore important de petites et moyennes erreurs subsistent. N'hésitez pas à me le signaler si vous en rencontrez.

J'ai bien quelques idées en tête pour poursuivre le développement ou améliorer certains points mais il est peu probable que je m'y intéresse rapidement.

Finalement, ce document regroupe les différentes pages d'aide écrite pour cette librairie. En espérant que cette dernière puisse vous être utile...

# PrettyCurses

## NOM

PrettyCurses

## SYNOPSIS

```
use PrettyCurses;
```

## DESCRIPTION

« **PrettyCurses** » est une librairie qui implémente principalement quelques widgets standard et qui en permet une gestion relativement simple et complète.

« **PrettyCurses** » introduit son propre type de variables pour les fenêtres. Il s'agit simplement d'un *Curses(3)* muni d'une donnée supplémentaire qui est la taille de la bordure. L'intérêt de cela est que les fonctions d'affichage n'écrivent pas par défaut sur cette bordure lorsque le texte est trop long pour tenir sur la fenêtre, ou que les barres de défilement s'affichent correctement.

## Variables globales

« **PrettyCurses** » définit un nombre assez important de variables globales qui peuvent ensuite être appelées n'importe où dans le programme. Attention, « **PrettyCurses** » n'initialise pas ces variables, voir à ce propos *PrettyCurses::default(3)*.

Voici la liste des variables globales :

### **\$PC\_stdscr**

Fenêtre qui correspond à l'écran standard. Exceptionnellement, cette variable est initialisée lors de l'appel de la fonction **PC\_initscr** et remise à jour lors de l'appel de la fonction **PC\_init**.

### **\$PC\_lower**

Chaîne de caractères regroupant les lettres en minuscules.

### **\$PC\_upper**

Chaîne de caractères regroupant les lettres majuscules correspondant aux lettres minuscules précédentes.

### **\$PC\_order**

Chaîne de caractères expliquant par quelle lettre il faut remplacer les lettres précédentes lorsque l'on désire les classer. Par exemple si « **é** » de **\$PC\_lower** correspond à « **e** » de **\$PC\_order**, la fonction **PC\_sort** classera les « **é** » avec les « **e** » et non à la fin comme le fait la fonction **sort** quand elle est mal configurée.

### **\$PC\_re\_lower**

Expression régulière qui reconnaît une lettre en minuscule.

### **\$PC\_re\_upper**

Expression régulière qui reconnaît une lettre en majuscule.

### **\$PC\_re\_letter**

Expression régulière qui reconnaît une lettre soit en minuscule, soit en majuscule.

### **\$PC\_re\_space**

Expression régulière qui reconnaît une espace ou un caractère équivalent.

### **\$PC\_re\_word**

Expression régulière qui reconnaît une lettre (ou un caractère) d'un mot. Attention, c'est ainsi l'expression régulière **/\$PC\_re\_word+/** qui reconnaît un mot.

`$PC_re_wordp`

Expression régulière qui reconnaît une lettre (ou un caractère) d'un mot pouvant être une abréviation se terminant pour un point (e.g. *Av.*).

`@PC_spacebeforepunct = ( qr/$regexp1/ => $string1, ... )`

Liste expliquant quel espacement il est nécessaire de mettre avant telle ponctuation. Attention, bien que cela se présente comme une table de hash, il s'agit bien d'une liste. Cela permet de garder un ordre dans les tests et donc de mettre des expressions régulières qui peuvent de recouper.

`@PC_spacebeforepunct = ( qr/$regexp1/ => $string1, ... )`

Liste expliquant quel espacement il est nécessaire de mettre après telle ponctuation.

`$PC_openquotes`

Chaîne de caractères définissant un guillemet ouvrant.

`$PC_closequotes`

Chaîne de caractères définissant un guillemet fermant.

`$PC_prefix_cmdtex`

Expression régulière qui reconnaît un caractère susceptible d'annoncer une commande `LATEX`. Bien que toutes les commandes `LATEX` sont introduites par un « `\` », cette variable globale n'est pas inutile lorsqu'il est nécessaire de repérer des erreurs de saisie.

`@PC_cmdtex = ( /$regexp1/ => [ /$regexp1.1/ => $string1.1, ... ], ... )`

Liste associant une référence sur une liste à une expression régulière censée reconnaître une certaine commande `LATEX`. La liste dont il est question précédemment associée à une expression régulière censée reconnaître l'argument de la commande `LATEX` dont il est question la chaîne de caractères par laquelle il va falloir remplacer cette commande. Ceci est notamment utile pour traiter les accents qui seraient tapés à la `LATEX` dans les champs de texte.

`$PC_re_mathtex`

Expression régulière reconnaissant une expression `LATEX` valide en mode maths.

`$PC_re_tex`

Expression régulière reconnaissant une expression `LATEX` valide.

`$PC_directory_default`

Format par défaut que doit utiliser la librairie `PrettyCurses::directory(3)`.

`$PC_lengthline`

Longueur par défaut d'une ligne.

`$PC_re0, ..., $PC_re9`

Variables supplémentaires supposées contenir des expressions régulières devant être réutilisées.

## Méthodes

### Initialisation

`PC_initscr ();`

Initialise `Curses(3)` et la variable globale `$PC_stdscr`. Cette fonction ne doit normalement être appelée qu'une unique fois au début du programme.

`PC_init ();`

Met à jour la variable globale `$PC_stdscr` et détruit toutes les autres fenêtres qui ont été créées jusqu'alors.

### Attendre un caractère

`$PC_win->PC_getch ();`

Attend un caractère sur la fenêtre `$PC_win`. Cette fonction reconnaît les séquences de caractères

tères que l'on lui a dit de reconnaître, ce qui sert notamment à intercepter des touches comme **SHIFT-LEFT** ou **CONTROL-F2**.

```
PC_getch_addsequences ( $sequence1 => $code1, ... );
```

Définit les séquences de caractères passées en arguments. Par exemple, l'appel de la fonction `PC_getch_addsequences ( "^[A" => "UP" );` permet de reconnaître une pression sur la touche `<UP>` et de renvoyer dans ce cas le code `UP`.

```
$PC_win->PC_getch_add ( $key1, ... );
```

Simule sur la fenêtre `$PC_win` la pression de la touche `$key1`, puis celle de la touche `$key2` et ainsi de suite.

```
$PC_win->PC_getch_on_press ( $code );
```

Définit le code à exécuter lorsqu'une touche est pressée dans l'écran `$PC_win` ou un de ses fils (voir la fonction `derwin`). La touche pressée est passée en argument au code.

## Fonctions usuelles de gestion d'une fenêtre

```
$PC_win->PC_getxy ( );
```

Renvoie la taille de la fenêtre `$PC_win` sous le format `($Y, $X)`.

```
$PC_win->PC_derwin ( $height_y, $height_x, $y, $x, $border, $write_on_border );
```

Crée une nouvelle fenêtre. Les deux premiers arguments `$height_y` et `$height_x` fournissent la taille de cette nouvelle fenêtre. Les deux arguments suivants `$y` et `$x` donnent la position du coin supérieur gauche de la fenêtre, position repérée par rapport à la fenêtre `$PC_win`. `$border` correspond à la taille de la bordure de la fenêtre en cours de création. `$write_on_border` est un booléen. S'il est positionné à une valeur définie et non nulle, la fenêtre nouvelle créée pourra empiéter sur la bordure de `$PC_win`. Finalement, cette fonction renvoie la fenêtre créée. Il se peut que les dimensions de celle-ci ne soient pas celles demandées si la place venait à manquer. Si encore pour faute de place, la fenêtre n'a pu être créée, la fonction renvoie un objet vide mais du bon type.

```
$PC_win->PC_clear ( );
```

Efface tout ce qu'il y a écrit sur la fenêtre `$PC_win`.

```
$PC_win->PC_border ( );
```

Dessine la bordure de la fenêtre `$PC_win`.

```
$PC_win->PC_defilbar ( $begin, $end, $button );
```

Dessine une barre de défilement sur la bordure de la fenêtre `$PC_win`. Ne fait rien si cette fenêtre n'a pas de bordure. `$begin` et `$end` fournissent respectivement le numéro de la première ligne affichée et celui de la dernière ligne affichée. `$button`, quant à lui, donne le nombre total de lignes. Il est conseillé de faire d'abord appel à `PC_border` avant afin d'effacer l'ancienne barre de défilement s'il y avait.

```
$PC_win->PC_addstr ( $y, $x, $s, $write_on_border );
```

Affiche sur la fenêtre `$PC_win`, à la position repérée par `$y` et `$x`, la chaîne de caractères `$s`. N'affiche pas les caractères qui débordent de la fenêtre. Le booléen `$write_on_border` précise s'il faut écrire sur la bordure ou s'il ne faut pas.

```
$PC_win->PC_refresh ( );
```

Dessine effectivement les modifications qui ont été faites dans la fenêtre `$PC_win`.

```
$PC_win->PC_delwin ( );
```

Supprime la fenêtre `$PC_win`. Attention, cela ne l'efface pas du tout de l'écran même après tous les appels à `$PC_refresh` possibles et imaginables; il est nécessaire de réécrire par dessus pour avoir cet effet.

## Gestion des couleurs

« `PrettyCurses` » redéfinit également ce qu'est une couleur (trop fort ;-). Une couleur, disons `$color`, est une référence sur une table de hash de la forme suivante :

```
$color = { NOFOCUS_FORE    =>    ,
           NOFOCUS_BACK   =>    ,
           FOCUS_FORE     =>    ,
           FOCUS_BACK     =>    };
```

Les valeurs affectées dans cette table de hash sont des noms de couleurs, tout ce qu'il y a de plus classique (e.g. `blue`, `white`...). `FORE` et `BACK` correspondent respectivement à la couleur du texte et à la couleur de fond.

Les fonctions qui permettent de gérer tout cela sont les suivantes :

```
PC_definecolors ( $name1 => $color1, ... );
```

Définit la couleur `$name1` et lui assigne la valeur `$color1` et ainsi de suite. Bien entendu, les `$color` doivent être des objets du type décrit précédemment.

```
$PC_win->PC_colorset ($color, $focus);
```

Déclare que désormais toutes les modifications sur la fenêtre `$PC_win` seront faites avec la couleur `$color`. Si `$focus` est positionné à une valeur définie et non nulle, utilise les couleurs définies par `FOCUS_*`, sinon utilise les couleurs définies par `NOFOCUS_*`. Si une ou plusieurs couleurs ne sont pas définies, remplace icelle par la couleur correspondante de `default`.

## Gestion du redimensionnement

« `PrettyCurses` » intercepte le signal `WINCH` émis lorsque la fenêtre en cours est redimensionnée. Il simule alors la pression d'une touche fictive nommée `WINCH` que le programme devra intercepter et gérer correctement, principalement en appelant la fonction `PC_init` et en recréant et redessinant toutes les fenêtres.

# PrettyCurses::default\_french

## NOM

PrettyCurses::default\_french

## SYNOPSIS

```
use PrettyCurses::default_french;
```

## DESCRIPTION

Initialise les variables globales définies par *PrettyCurses(3)* en tenant plus ou moins compte des normes de la typographie française.

Définit en outre les séquences d'échappement classiques et les couleurs `default`, `fields` et `error`.

# PrettyCurses::utils

## NOM

PrettyCurses::utils

## SYNOPSIS

```
use PrettyCurses::utils;

PC_upper ($text);
PC_lower ($text);
PC_truncate ($text, $length);

PC_compare ($a, $b);
PC_sort (@list);

PC_copy ($a);
PC_equal ($a, $b);

PC_addorder ($new, $old);
```

## DESCRIPTION

Définit quelques fonctions parfois bien utiles. En voici la liste :

### Gestion des chaînes de caractères

**PC\_upper (\$text);**

Met en majuscules la chaîne de caractères **\$text** en utilisant la correspondance donnée par les variables globales **\$PC\_lower** et **\$PC\_upper** et renvoie le résultat.

**PC\_lower (\$text);**

Met en minuscules la chaîne de caractères **\$text** en utilisant la correspondance donnée par les variables globales **\$PC\_lower** et **\$PC\_upper** et renvoie le résultat.

**PC\_order (\$text);**

Renvoie ce qui permet de classer alphabétiquement la chaîne de caractères **\$text** en utilisant la correspondance donnée par les variables globales **\$PC\_lower**, **\$PC\_upper** et **\$PC\_order**.

**PC\_truncate (\$text, \$length);**

Coupe le texte **\$text** pour qu'il tienne sur des lignes de longueur **\$length**. Les « \n » déjà présents sont conservés. Si le paramètre **\$length** n'est pas défini, la valeur utilisée est celle de la variable globale **\$PC\_lengthline**. Renvoie une référence sur la liste des lignes.

### Fonctions de comparaison

**PC\_compare (\$a, \$b);**

Compare les chaînes de caractères **\$a** et **\$b** en utilisant l'ordre défini par la variable globale **\$PC\_order**. Renvoie -1 si **\$a** est plus petit que **\$b**, 0 s'ils sont égaux et 1 si **\$a** est plus grand que **\$b**.

**PC\_sort (@list);**

Trie la liste **@list** en utilisant l'ordre défini par la variable globale **\$PC\_order** et renvoie la liste triée.

## Gestion des structures complexes

`PC_copy ($a);`

Renvoie une copie de l'objet `$a`. Attention, cette fonction ne fonctionne correctement que si l'objet `$a` ne contient récursivement que des références sur des `LIST` et sur des `HASH`.

`PC_equal ($a, $b);`

Compare les objets `$a` et `$b`. Là encore, cette fonction ne fonctionne correctement que si les objets `$a` et `$b` ne font intervenir récursivement que des références sur des `LIST` et sur des `HASH`. Renvoie 1 si les objets sont égaux, 0 sinon.

## Autre fonction

`PC_addorder ($new, $old);`

Forme un ordre composé à mettre dans une table de hash `KEYS` (voir *PrettyCurses::Widgets(3)*) à partir de `$new` et de `$old`. L'ordre qui en résulte est de la forme `$new/$old`. Si toutefois `$old` était un ordre prioritaire, `$new` n'est pas ajouté.

# PrettyCurses::Message

## NOM

PrettyCurses::Message

## SYNOPSIS

```
use PrettyCurses::Message;

PC_message ($PC_win, $param);
```

## DESCRIPTION

Affiche un message sur la fenêtre `$PC_win`. Le message et son format sont fournis grâce à l'argument `$param`. Il s'agit d'une référence sur une table de hash dont le format est le suivant :

```
$param = {   HEIGHT_X           =>    ,
            MESSAGE_TEXT       =>    ,
            BUTTON_TEXT        =>    ,
            KEYS                =>    };
```

Détaillons ces paramètres.

### HEIGHT\_X

Définit la taille horizontale de la fenêtre du message. Si ce paramètre n'est pas fourni, la taille prise par défaut correspond aux deux tiers de la largeur de la fenêtre `$PC_win`.

### MESSAGE\_TEXT

Texte du message. Il peut s'agir soit d'une chaîne de caractères, soit d'une référence sur une liste. Dans le deuxième cas, tous les éléments de la liste sont affichés dans l'ordre, un saut de ligne est inséré entre chacun d'eux. Il est tout à fait possible de mettre des « `\n` » dans ces chaînes de caractères, ils seront interprétés comme des retours à la ligne. Finalement, il n'est pas nécessaire de faire attention à formater le texte pour qu'il ne dépasse pas en largeur de la fenêtre, ceci est fait automatiquement.

### BUTTON\_TEXT

Texte qui doit apparaître sur le bouton au bas du message. La valeur par défaut est « `OK` ».

**KEYS** Voir la description générale de *PrettyCurses::Widgets(3)*.

# PrettyCurses::Corrections

## NOM

PrettyCurses::Corrections

## SYNOPSIS

```
use PrettyCurses::Corrections;

PrettyCurses::Corrections::tex ($value);

PrettyCurses::Corrections::quotes ($value);
PrettyCurses::Corrections::punctuation ($value);

PrettyCurses::Corrections::caps ($value);
PrettyCurses::Corrections::small_caps ($value);
PrettyCurses::Corrections::transliteration ($value);

PrettyCurses::Corrections::telephone ($value);
```

## DESCRIPTION

Ce package implémente un certain nombre de fonctions adhoc pour effectuer les corrections automatiques. Il est utilisé directement par *PrettyCurses::TextField(3)* et donc aussi indirectement par plusieurs autres packages.

L'argument `$value` doit être une référence sur une liste de références de listes de chaînes de caractères comme cela est détaillé dans *PrettyCurses::TextField(3)*. La valeur de retour de chacune de ces fonctions est un objet du même type. Il faut noter que justement cette structure permet de ne jamais perdre la valeur initiale et de proposer ensuite à l'utilisateur d'accepter ou de refuser la correction proposée.

Voici la liste des fonctions en question :

### **L<sup>A</sup>T<sub>E</sub>X**

```
PrettyCurses::Corrections::tex ($value);
```

Oublie les anciennes corrections. Vérifie si la valeur passée est une expression L<sup>A</sup>T<sub>E</sub>X valide selon l'expression régulière de la variable globale `$PC_re_tex` et si c'est le cas, reconnaît les commandes L<sup>A</sup>T<sub>E</sub>X présentes dans la variable globale `@PC_cmdtex` et les remplace par les valeurs correspondantes.

### **Typographie**

```
PrettyCurses::Corrections::quotes ($value);
```

Repère les guillemets et les remplace par le standard défini dans les variables globales `$PC_openquotes` et `$PC_closequotes`.

```
PrettyCurses::Corrections::punctuation ($value);
```

Corrige l'espacement autour des ponctuations.

Les fonctions précédentes n'ont pas toujours un comportement très à propos. Afin de limiter les dégats, il est conseillé de les appeler dans l'ordre dans lequel elles ont été présentées. C'est exactement ce que fait *PrettyCurses::TextField(3)* sur appel de la fonction `correct`.

## Correction sur les mots

`PrettyCurses::Corrections::caps ($value);`

Oublie les anciennes corrections. Écrit tous les mots en majuscules.

`PrettyCurses::Corrections::small_caps ($value);`

Oublie les anciennes corrections. Écrit tous les mots en petites majuscules (ie écrit la première lettre en majuscules et les autres lettres en minuscules).

`PrettyCurses::Corrections::transliteration ($value, $tr, $punct);`

L'argument `$tr` est une référence sur une liste qui met en correspondance des expressions régulières avec des chaînes de caractères. La fonction commence par oublier les anciennes corrections puis regarde si chacun des mots de `$value` est reconnu par les expressions régulières de la liste `$tr`. Si c'est le cas, la fonction remplace ce mot par la valeur qui est associée à cette expression régulière. Cette valeur peut contenir des `$1`, `$2`, etc. Ils seront interprétés correctement sauf s'ils sont protégés par des « `\` ». Le booléen `$punct` précise s'il faut utiliser l'expression régulière `/$PC_word+/` pour reconnaître un mot (cas `$punct = 1`) ou s'il faut utiliser l'expression régulière `/$PC_wordp+/` (cas `$punct = 0`).

# PrettyCurses::Widgets

## NOM

PrettyCurses::Widgets

## SYNOPSIS

```
use PrettyCurses::<widget>;

$w = PrettyCurses::<widget>->new ({
    X           => ,
    Y           => ,
    HEIGHT_X    => ,
    HEIGHT_Y    => ,
    VALUE       => ,
    KEYS        => ,
    READONLY    => ,
    UNDO        => ,
    CURRENT_LINE => ,
    CURRENT_FIELD => ,
    CURSOR_POS  => ,
    CORRECT     => ,
    COLOR       => ,
    COLOR_ERR   => ,
    DRAW_AFTER_EXECUTE =>
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );
$w->getValue ();
$w->setValue ($value);

$w->draw ($PC_win, $focus);
$w->execute ($PC_win);

$w->correct ($PC_win, $focus);

$w->undo ($PC_win, $focus);
$w->redo ($PC_win, $focus);
$w->save_position ($name);
$w->remove_position ($name);
$w->addinhistory ($name);
$w->clearhistory ();
```

## DESCRIPTION

Cette page ne décrit pas à proprement parler un package, mais plutôt la syntaxe générale de tous les widgets proposés par *PrettyCurses(3)*. Bien que cela ne soit pas programmé comme ça, il est bon de voir cela comme une classe abstraite dont les widgets dérivent.

Les widgets en question sont `Button`, `Caption`, `CheckBox`, `Form`, `Menu`, `MultiFields`, `TextField` et `TextMemo`.

## Créer une nouvelle instance d'un widget

Cela se fait grâce à la fonction `new` dont la syntaxe générale a été décrite précédemment. Nous allons décrire un par un ce à quoi correspondent les paramètres cités précédemment. Ceux-ci sont ceux qui sont censés être définis pour tous les widgets. Bien sûr, certains paramètres n'ont pas de sens pour certains widgets (e.g. `READONLY` pour un widget `Caption`) mais cela n'est pas grave. Moralement, ce que l'on a écrit impose que si un widget utilise un des paramètres donnés ci-dessus, il devra lui conférer le sens que l'on va décrire tout de suite.

Finalement, si un de ces paramètres n'est pas défini lors de l'appel de la fonction `new`, il est généralement positionné à une valeur par défaut. Toutefois, celle-ci dépend du widget dont on veut créer une instance. Les valeurs par défaut ne sont pas décrites ici mais dans les pages de mân spécifiques.

**X et Y**

Décrivent la position du widget sur la fenêtre sur laquelle il va être afficher.

**HEIGHT\_X et HEIGHT\_Y**

Décrivent la taille du widget.

**VALUE**

Valeur du widget.

**KEYS** Il s'agit d'une référence sur une table de hash qui précise le comportement à avoir lorsqu'une certaine touche, disons `$key`, a été enfoncée pendant l'exécution du widget. Si `$key` n'apparaît pas dans les clés de la table de hash, la touche est ignorée. Si au contraire, `$key` apparaît, il y a deux cas : soit la valeur qui lui est associée est une référence sur un `CODE` et alors ce code est exécuté recevant comme paramètre une référence sur le widget en question et l'exécution se poursuit normalement, soit ce n'est pas le cas et alors l'exécution du widget s'arrête et renvoie la valeur associée. Si le widget, pour une raison quelconque, devait intercepter précisément la touche `$key`, alors cette interception est prioritaire sauf dans les deux cas suivants : la valeur associée est une référence sur un `CODE` ou ce n'est pas le cas et celle-ci débute par le caractère « / ». Il y a deux codes particuliers qui sont `ON_PRESS` et `ON_MODIFY`. Ils ne sont prioritaires sur aucune autre interception et sont exécutés respectivement lorsqu'une quelconque touche a été enfoncée et lorsque le widget a été modifié. Bien sûr, `ON_MODIFY` est prioritaire sur `ON_PRESS`.

**READONLY**

Précise si le widget est modifiable (cas `READONLY => 0`) ou si ce n'est pas le cas (cas `READONLY => 1`).

**UNDO** Précise s'il faut retenir l'historique et si les touches `CONTROL_PGUP` et `CONTROL_PGDOWN` doivent respectivement annuler et refaire la dernière modification (cas `UNDO => 1`) ou si ce n'est pas nécessaire (cas `UNDO => 0`).

**CURRENT\_LINE**

Numéro de la ligne courante. Ce paramètre peut être interprété différemment selon les widgets, il est plus sûr de se reporter aux pages de mân spécifiques pour connaître son comportement exact.

**CURRENT\_FIELD**

Numéro du champ courant si le widget comprend plusieurs champs. Ce paramètre peut être interprété différemment selon les widgets, il est plus sûr de se reporter aux pages de mân spécifiques pour connaître son comportement exact.

**CURSOR\_POS**

Position du curseur. Ce paramètre peut être interprété différemment selon les widgets, il est plus sûr de se reporter aux pages de mân spécifiques pour connaître son comportement exact.

**CORRECT**

Précise s'il faut faire les corrections automatiques lorsque la combinaison de touches `CONTROL-T` est enfoncée ou si ce n'est pas nécessaire.

## COLOR

Couleur avec laquelle devra être dessinée le widget.

## COLOR\_ERR

Couleur avec laquelle seront dessinées les parties erronées du widget. Voir *PrettyCurses::TextField(3)* pour plus d'informations.

## DRAW\_AFTER\_EXECUTE

Précise s'il faut redessiner le widget après l'exécution (cas `DRAW_AFTER_EXECUTE => 1`) ou si ce n'est pas nécessaire (cas `DRAW_AFTER_EXECUTE => 0`).

## Méthodes

### Gestion des paramètres

```
$w->getParam ($name);
```

Renvoie la valeur du paramètre `$name`.

```
$w->setParam ( $name1 => $value1, ... );
```

Affecte la valeur `$value1` au paramètre `$name1` et ainsi de suite.

```
$w->getValue ();
```

Renvoie la valeur du paramètre `VALUE`. Attention, cela n'est pas forcément équivalent à la commande `$w->getParam ('VALUE')`; parfois les paramètres de ce type étant stockés de façon spéciale. Pour plus d'informations, se reporter aux pages de man spécifiques.

```
$w->setValue ($value);
```

Met à jour la valeur du paramètre `VALUE`. Même remarque que précédemment.

### Fonctions d'affichage

```
$w->draw ($PC_win, $focus);
```

Dessine le widget `$w` sur la fenêtre `$PC_win` puis rend la main. Si `$focus` est positionné à une valeur définie et non nulle, dessine le widget comme si celui-ci avait le focus.

```
$w->execute ($PC_win);
```

Lance l'exécution du widget `$w`. Comme expliqué précédemment, la table de hash définie par le paramètre `KEYS` détermine lorsque cette procédure termine et le code de retour renvoyé. Redessine le widget via la commande `$w->draw ($PC_win)`; avant de rendre la main si le paramètre `DRAW_AFTER_EXECUTE` est positionné à 1. Il est par exemple intéressant de positionner cette valeur à 0 lorsque les événements `ON_PRESS` ou `ON_MODIFY` sont interceptés.

### Autres fonctions

```
$w->correct ($PC_win, $focus);
```

Fait les corrections automatiques sur la valeur du widget `$w` (voir les pages de man spécifiques et *PrettyCurses::Corrections(3)* pour plus d'informations). Si `$PC_win` est défini, redessine finalement le widget `$w` via la commande `$w->draw ($PC_win, $focus)`; Si le paramètre `CORRECT` est positionné à 1, cette fonction est appelée automatiquement lorsque la combinaison de touches `CONTROL-T` est pressée.

```
$w->undo ($PC_win, $focus);
```

Remonte d'un cran dans l'historique. Si `$PC_win` est défini, redessine finalement le widget `$w` via la commande `$w->draw ($PC_win, $focus)`; Si le paramètre `UNDO` est positionné à 1, cette fonction est appelée automatiquement lorsque la combinaison de touches `CONTROL-PGUP` est pressée.

`$w->redo ($PC_win, $focus);`

Descend d'un cran dans l'historique. Si `$PC_win` est défini, redessine finalement le widget `$w` via la commande `$w->draw ($PC_win, $focus);`. Si le paramètre `UNDO` est positionné à 1, cette fonction est appelée automatiquement lorsque la combinaison de touches `CONTROL-PGDOWN` est pressée.

`$w->save_position ($name);`

Sauvegarde la position actuelle sous le nom `$name`. Quelques noms sont utilisés par la librairie elle-même. Tous ceux-ci sont introduits par la chaîne de caractères « `__` ». Il est donc déconseillé d'utiliser des noms commençant par « `__` ».

`$w->unlock_position ($name);`

Supprime la position `$name`.

`$w->addinhistory ($name);`

Ajoute dans l'historique la position `$name`. Si aucun argument n'est passé, y ajoute la position actuelle. Si le paramètre `UNDO` est positionné à 1, cette fonction est appelée automatiquement *avant* chaque modification du widget.

`$w->clearhistory ();`

Efface l'historique.

# PrettyCurses::Caption

## NOM

PrettyCurses::Caption

## SYNOPSIS

```
use PrettyCurses::Caption;

$w = PrettyCurses::Caption->new ({
    X           => 0,
    Y           => 0,
    HEIGHT_X    => 10,
    HEIGHT_Y    => 1,
    VALUE       => '',
    COLOR       => 'default',
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );
$w->getValue ();
$w->setValue ($value);

$w->draw ($PC_win);
```

## DESCRIPTION

Ce package implémente un widget Caption (légende).

## Paramètres

Pour les descriptions générales, se reporter à la page *PrettyCurses::Widgets(3)*. Les valeurs qui sont données ci-dessus sont celles qui sont passés par défaut si elles ne sont pas initialisées lors de l'appel de la fonction `new`.

Il faut signaler que le paramètre `VALUE` peut-être soit une référence sur une liste, soit une chaîne de caractères. Si c'est une référence sur une liste, chaque élément de la liste est dessinée et un retour à la ligne est insérée entre chacun d'eux. Finalement les caractères « `\n` » sont autorisés et sont compris naturellement comme des retours à la ligne.

## Méthodes

Se reporter à la page *PrettyCurses::Widgets(3)*.

# PrettyCurses::TextField

## NOM

PrettyCurses::TextField

## SYNOPSIS

```
use PrettyCurses::TextField;

$w = PrettyCurses::TextField->new ({
    X           => 0,
    Y           => 0,
    HEIGHT_X    => 10,
    LENGTH      => 0,
    VALUE       => [ ],
    KEYS        => { },
    READONLY    => 0,
    UNDO        => 1,
    CURSOR_POS  => 0,
    CORRECT     => 1,
    COLOR       => 'fields',
    COLOR_ERR   => 'error',
    DRAW_AFTER_EXECUTE => 1
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );
$w->getValue ();
$w->setValue ($value);

$w->draw ($PC_win);
$w->execute ($PC_win);

$w->correct ($PC_win, $focus);

$w->undo ($PC_win, $focus);
$w->redo ($PC_win, $focus);
$w->save_position ($name);
$w->remove_position ($name);
$w->addinhistory ($name);
$w->clearhistory ();
```

## DESCRIPTION

Ce package implémente un widget TextField (champ de texte).

## Paramètres

Ici ne sont détaillés que les paramètres spécifiques au widget TextField. Pour les descriptions générales, se reporter à la page *PrettyCurses::Widgets(3)*. Les valeurs qui sont données ci-dessus sont celles qui sont passés par défaut si elles ne sont pas initialisées lors de l'appel de la fonction `new`.

## LENGTH

Définit le nombre maximal de caractères que peut contenir le champ. Si ce paramètre est positionné à 0, aucune limite n'est fixée.

## VALUE

Voir le paragraphe suivant.

## CURSOR\_POS

Position du curseur sur la ligne. Un nombre négatif indique que la position doit être comptée à partir de la fin du champ.

## Description du paramètre VALUE

Ce paramètre contient la valeur du champ. Il est toutefois probablement nécessaire de préciser le format. Il s'agit d'une référence sur une liste, chaque élément de cette liste étant lui-même une référence sur une liste, cette dernière référence définissant une partie du champ de la manière suivante. Cette liste contient les chaînes de caractères susceptibles d'apparaître dans la valeur du champ, celle qui apparaît effectivement étant la première de la liste.

Cela sera peut-être plus clair sur un exemple. Si VALUE est positionné à :

```
[ [ "Bonjour", "Hello" ], [ " Xavier !" ] ]
```

la valeur du champ à cet instant sera « **Bonjour Xavier !** », mais il sera possible de passer rapidement à « **Hello Xavier !** » (grâce à la combinaison de touches **CONTROL-A**) ; VALUE sera alors positionné à :

```
[ [ "Hello", "Bonjour" ], [ " Xavier !" ] ]
```

de sorte qu'il est encore possible de revenir rapidement à l'ancienne valeur.

Les parties de champ pour lesquelles plusieurs possibilités sont offertes sont affichées avec la couleur définie par le paramètre **COLOR\_ERR**, alors que les autres sont affichées avec la couleur définie par le paramètre **COLOR**.

Finalement, l'appel de la fonction `$w->setValue ($value)` ; affecte la valeur `[ [ $value ] ]` au paramètre VALUE tandis que la fonction `$w->getValue ()` ; renvoie la chaîne de caractères correspondante à la valeur courante. Pour être plus précis, il est nécessaire d'utiliser les fonctions `setParam` et `getParam`.

## Méthodes

Se reporter à la page *PrettyCurses::Widgets(3)*.

## Touches interceptées lors de l'exécution

### LEFT, RIGHT, HOME, END

Se déplace à l'endroit voulu dans le champ sauf si cela n'est pas possible auquel cas la touche n'est pas interceptée.

### SHIFT-LEFT

Se déplace au début du mot où la partie de champ courant. Si le curseur est déjà au début du champ, la touche n'est pas interceptée.

### SHIFT-RIGHT

Se déplace au début du mot suivant ou à la fin de la partie de champ courante. Si le curseur est déjà à la fin du champ, la touche n'est pas interceptée.

### DELETE, BACKSPACE

Efface le caractère attendu. Si cela n'est pas possible, la touche n'est pas interceptée. Ces touches ne sont de toute façon pas prises en compte si le paramètre **READONLY** est positionné à 1.

#### INSERT

Bascule entre le mode insertion et le mode remplacement.

#### CONTROL-U

Efface tout le champ sauf si le champ est déjà vide auquel cas la touche n'est pas interceptée. Cette touche n'est de toute façon pas prise en compte si le paramètre **READONLY** est positionné à 1.

#### CONTROL-W

Efface le mot précédent. Cette touche n'est de toute façon pas prise en compte si le paramètre **READONLY** est positionné à 1.

#### CONTROL-A

Bascule la valeur effective de la partie de champ courante. Cette touche n'est de toute façon pas prise en compte si le paramètre **READONLY** est positionné à 1.

#### CONTROL-T

Applique les corrections automatiques. Cette touche n'est pas prise en compte si le paramètre **CORRECT** est positionné à 0 ou si le paramètre **READONLY** est positionné à 1.

#### CONTROL-PGUP

Remonte d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà au début de l'historique. Cette touche est de toute façon ignorée si le paramètre **UNDO** est positionné à 0 ou si le paramètre **READONLY** est positionné à 1.

#### CONTROL-PGDOWN

Descend d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà à la fin de l'historique. Cette touche est de toute façon ignorée si le paramètre **UNDO** est positionné à 0 ou si le paramètre **READONLY** est positionné à 1.

# PrettyCurses::TextMemo

## NOM

PrettyCurses::TextMemo

## SYNOPSIS

```
use PrettyCurses::TextMemo;

$w = PrettyCurses::TextMemo->new ({
    X           => 0,
    Y           => 0,
    HEIGHT_X    => 10,
    HEIGHT_Y    => 2,
    VALUE       => [ ],
    KEYS        => { },
    READONLY    => 0,
    UNDO        => 1,
    CURRENT_LINE => 0,
    CURSOR_POS  => 0,
    CORRECT     => 1,
    COLOR       => 'fields',
    COLOR_ERR   => 'error',
    DRAW_AFTER_EXECUTE => 1
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );
$w->getValue ();
$w->setValue ($value);

$w->draw ($PC_win);
$w->execute ($PC_win);

$w->correct ($PC_win, $focus);

$w->undo ($PC_win, $focus);
$w->redo ($PC_win, $focus);
$w->save_position ($name);
$w->remove_position ($name);
$w->addinhistory ($name);
$w->clearhistory ();
```

## DESCRIPTION

Ce package implémente un widget TextMemo. Il s'agit d'un champ de texte pouvant tenir sur un nombre arbitraire de lignes.

## Paramètres

Ici ne sont détaillés que les paramètres spécifiques au widget `TextMemo`. Pour les descriptions générales, se reporter à la page *`PrettyCurses::Widgets(3)`*. Les valeurs qui sont données ci-dessus sont celles qui sont passés par défaut si elles ne sont pas initialisées lors de l'appel de la fonction `new`.

### VALUE

Référence sur une liste dont les éléments sont des objets de type celui du paramètre `VALUE` de *`PrettyCurses::TextField(3)`*. Bien sûr, chaque élément correspond à une ligne du champ. Comme dans *`PrettyCurses::TextField(3)`*, les fonctions `getValue` et `setValue` ont un comportement singulier. Ici, la fonction `getValue` renvoie juste une liste de chaînes de caractères et la fonction `setValue` prend une telle liste et l'affecte correctement au paramètre `VALUE`.

### CURRENT\_LINE

Ligne sur laquelle se trouve le curseur. Si ce nombre est négatif, les lignes sont comptées à partir de la dernière.

### CURSOR\_POS

Position du curseur sur la ligne déterminée par `CURRENT_LINE`. Si ce nombre est négatif, la position est comptée à partir de la fin de la ligne.

## Méthodes

Se reporter à la page *`PrettyCurses::Widgets(3)`*.

## Touches interceptées lors de l'exécution

Il faut noter que *`PrettyCurses::TextMemo(3)`* fait appel à *`PrettyCurses::TextField(3)`* pour gérer séparément chaque ligne de champ et que les interceptions de *`PrettyCurses::TextField(3)`* sont en général prioritaires. Expliquons cela en regardant par exemple le comportement de la touche `LEFT`. Si le curseur ne se trouve pas au début d'une ligne, c'est *`PrettyCurses::TextField(3)`* qui intercepte la touche et déplace simplement ce curseur vers la gauche. Sinon, c'est *`PrettyCurses::TextMemo(3)`* qui intercepte la touche et le curseur remonte à la ligne précédente si elle existe.

**LEFT** Déplace le curseur à la fin de la ligne précédente. Si le curseur était sur la première ligne, la touche n'est pas interceptée.

### RIGHT

Déplace le curseur au début de la ligne suivante. Si le curseur était sur la dernière ligne, la touche n'est pas interceptée.

**UP** Accède à la ligne précédente. Si le curseur était déjà positionné sur la première ligne, la touche n'est pas interceptée.

**DOWN** Accède à la ligne suivante. Si le curseur était déjà positionné sur la dernière ligne, la touche n'est pas interceptée.

### BACKSPACE

Réunit la ligne en cours avec la ligne précédente. Cette touche n'est pas prise en compte si le paramètre `READONLY` est positionné à 1.

### DELETE

Réunit la ligne en cours avec la ligne suivante. Cette touche n'est pas prise en compte si le paramètre `READONLY` est positionné à 1.

### ENTER

Sépare la ligne en cours en deux lignes. Cette touche n'est pas interceptée si le curseur est positionné sur la dernière ligne du champ et que celle-ci est vide. Cette touche n'est de toute façon pas prise en compte si le paramètre `READONLY` est positionné à 1.

#### CONTROL-U

Supprime la ligne en cours. Cette touche n'est pas prise en compte si le paramètre **READONLY** est positionné à 1.

#### CONTROL-T

Applique les corrections automatiques. Cette touche n'est pas prise en compte si le paramètre **CORRECT** est positionné à 0 ou si le paramètre **READONLY** est positionné à 1. Il faut noter que les *PrettyCurses::TextField(3)* utilisés sont appelés avec le paramètre **CORRECT** positionné à 0 de sorte qu'ils ne peuvent pas intercepter cette touche.

#### CONTROL-PGUP

Remonte d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà au début de l'historique. Cette touche est de toute façon ignorée si le paramètre **UNDO** est positionné à 0 ou si le paramètre **READONLY** est positionné à 1.

#### CONTROL-PGDOWN

Descend d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà à la fin de l'historique. Cette touche est de toute façon ignorée si le paramètre **UNDO** est positionné à 0 ou si le paramètre **READONLY** est positionné à 1.

# PrettyCurses::MultiFields

## NOM

PrettyCurses::MultiFields

## SYNOPSIS

```
$w = PrettyCurses::MultiFields->new ({
    X           => 0,
    Y           => 0,
    HEIGHT_X    => 10,
    VALUE       => [ ],
    KEYS        => { },
    READONLY    => 0,
    UNDO        => 1,
    CURRENT_FIELD => 0,
    MAX_FIELDS  => 0,
    CURSOR_POS  => 0,
    CORRECT     => 1,
    COLOR       => 'fields',
    COLOR_ERR   => 'error',
    SEPARATOR   => '/',
    COLOR_SEP   => 'default',
    DRAW_AFTER_EXECUTE => 1
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );
$w->getValue ();
$w->setValue ($value);

$w->draw ($PC_win);
$w->execute ($PC_win);

$w->correct ($PC_win, $focus);

$w->undo ($PC_win, $focus);
$w->redo ($PC_win, $focus);
$w->save_position ($name);
$w->remove_position ($name);
$w->addinhistory ($name);
$w->clearhistory ();
```

## DESCRIPTION

Ce package implémente un widget MultiFields. Il s'agit d'une zone de texte pouvant contenir plusieurs champs à la suite. Ceci est notamment pratique pour faire saisir un chemin, chaque champ correspondra à un sous-répertoire.

## Paramètres

Ici ne sont détaillés que les paramètres spécifiques au widget `MultiFields`. Pour les descriptions générales, se reporter à la page *PrettyCurses::Widgets(3)*. Les valeurs qui sont données ci-dessus sont celles qui sont passés par défaut si elles ne sont pas initialisées lors de l'appel de la fonction `new`.

### VALUE

Référence sur une liste dont les éléments sont des objets de type celui du paramètre `VALUE` de *PrettyCurses::TextField(3)*. Bien sûr, chaque élément correspond à un champ. Comme dans *PrettyCurses::TextField(3)*, les fonctions `getValue` et `setValue` ont un comportement singulier. Ici, la fonction `getValue` renvoie juste une liste de chaînes de caractères et la fonction `setValue` prend une telle liste et l'affecte correctement au paramètre `VALUE`.

### CURRENT\_FIELD

Numéro du champ actif. Un nombre négatif précise que les champs sont comptés à partir du dernier.

### MAX\_FIELDS

Nombre maximum de champs. Le nombre de champs n'est pas limité si ce paramètre est positionné à 0. Toutefois, le nombre de champs est toujours limité par la taille du widget, en imposant à chaque champ de disposer d'au moins deux caractères.

### CURSOR\_POS

Position du curseur sur le champ actif. Un nombre négatif précise que cette position doit être comptée à partir de la droite du champ.

### SEPARATOR

Chaîne de caractères à introduire pour séparer deux champs.

### COLOR\_SEP

Couleur dans laquelle doit être dessinée la chaîne de caractères précédente.

## Méthodes

Se reporter à la page *PrettyCurses::Widgets(3)*.

## Touches interceptées lors de l'exécution

Il faut noter que *PrettyCurses::MultiFields(3)* fait appel à *PrettyCurses::TextField(3)* pour gérer séparément chaque ligne de champ et que les interceptions de *PrettyCurses::TextField(3)* sont en général prioritaires. Expliquons cela en regardant par exemple le comportement de la touche `LEFT`. Si le curseur ne se trouve pas au début d'une ligne, c'est *PrettyCurses::TextField(3)* qui intercepte la touche et déplace simplement ce curseur vers la gauche. Sinon, c'est *PrettyCurses::MultiFields(3)* qui intercepte la touche et le curseur est déplacé à la fin du champ précédent, le champ en cours est éventuellement supprimé s'il était vide.

### LEFT ou SHIFT-LEFT

Déplace le curseur à la fin du champ précédent. Efface le champ en cours si celui-ci était le dernier et vide et si le paramètre `READONLY` est positionné à 0. La touche n'est pas interceptée si le champ actif était le premier.

### RIGHT ou SHIFT-RIGHT

Déplace le curseur au début du champ suivant. Si le champ en cours était le dernier, un nouveau champ est créé à moins que le nombre maximal de champs soit déjà atteint ou que le paramètre `READONLY` soit positionné à 0. Dans ces derniers cas, la touche n'est pas interceptée.

### BACKSPACE

Regroupe le champ en cours avec le champ précédent. Cette touche n'est pas interceptée si le champ courant est le premier champ.

#### DELETE

Regroupe le champ en cours avec le champ suivant. Cette touche n'est pas interceptée si le champ courant est le dernier champ.

#### CONTROL-U

Efface tous les champs. Cette touche est ignorée si le paramètre `READONLY` est positionné à 1.

#### CONTROL-T

Applique les corrections automatiques. Cette touche n'est pas prise en compte si le paramètre `CORRECT` est positionné à 0 ou si le paramètre `READONLY` est positionné à 1. Il faut noter que les *PrettyCurses::TextField(3)* utilisés sont appelés avec le paramètre `CORRECT` positionné à 0 de sorte qu'ils ne peuvent pas intercepter cette touche.

#### CONTROL-PGUP

Remonte d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà au début de l'historique. Cette touche est de toute façon ignorée si le paramètre `UNDO` est positionné à 0 ou si le paramètre `READONLY` est positionné à 1.

#### CONTROL-PGDOWN

Descend d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà à la fin de l'historique. Cette touche est de toute façon ignorée si le paramètre `UNDO` est positionné à 0 ou si le paramètre `READONLY` est positionné à 1.

# PrettyCurses::Menu

## NOM

PrettyCurses::Menu

## SYNOPSIS

```
use PrettyCurses::Menu;

$w = PrettyCurses::Menu->new ({
    X           => 0,
    Y           => 0,
    HEIGHT_X    => 10,
    HEIGHT_Y    => 10,
    KEYS        => { },
    COLOR       => 'default',
    BORDER      => 1,
    DEFIL_BAR   => 1,
    DRAW_AFTER_EXECUTE => 1,

    VALUE       => ,
    TREE        => [ ],
    ITEMS       => ,
    LENGTHS     => [ ],
    HEAD_LINE   => 0,
    SELECTED_ITEM => -1,
    COLORS      => [ ],
    MODE_SCROLL => 1,

    ENABLE_SEARCH => 1,
    CURRENT_FIELD => -1,
    SEPARATOR    => '/',
    MAX_DEEP     => ,
    COLOR_FIELDS => 'fields',
    MODE_MODIFY  => 1
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );
$w->getValue ();

$w->draw ($PC_win, $focus);
$w->execute ($PC_win);
```

## DESCRIPTION

Ce package implémente un widget menu qui offre des possibilités de sous-menus intégrés.

Chaque entrée du menu contient un certain nombre de champs fixé par la taille de la liste `LENGTHS`. C'est pour cette raison que les paramètres `CAPTION`, `LENGTHS` et `COLORS` sont des références sur des listes. Ceci alourdit certes la syntaxe parfois de façon inutile mais peut parfois être bien pratique.

Donnons tout de suite un exemple :

```

perl                                rwx
|- librairies                       rwx
|  |- PrettyCurses.pm              rw-
|  \- PrettyCurses                 rwx
|     |- Button.pm                 rw-
|     |- Caption.pm                rw-
|     |- CheckBox.pm               rw-
|     |- Corrections.pm            rw-
|     |- default_french.pm         rw-
|     |- default.pm@               rwx
|     |- directory.pm              rw-
|     |- Form.pm                   rw-
|     |- Menu.pm                   rw-
|     |- Message.pm                rw-
|     |- MultiFields.pm            rw-
|     |- TextField.pm              rw-
|     |- TextMemo.pm               rw-
|     \- utils.pm                  rw-
\-- myprog.pl                       rw-

```

Ce menu est donc formé de deux champs : le premier correspond au nom du fichier et le deuxième aux droits que l'utilisateur a sur ce fichier. Ici `LENGTHS` pourra par exemple être initialisé à `[70, 3]`.

## Fonction de recherche

Ce menu dispose en outre d'une fonction de recherche. Celle-ci est matérialisée principalement par des champs de recherche, un par champ constituant une entrée. L'ensemble des valeurs de ces champs est appelé valeur de recherche.

Il est nécessaire de savoir lorsqu'une certaine valeur de recherche reconnaît une certaine entrée. Reprenons peut-être l'exemple précédent pour expliquer les choses. Chaque champ de recherche correspond à un champ du menu. Le premier est un `PrettyCurses::MultiFields(3)` dont chaque partie correspond à une sous-branche du menu. Les autres sont des `PrettyCurses::TextField(3)`. Pour qu'un certain texte soit reconnu par un champ de recherche, il faut que celui-ci commence par la valeur de ce champ. Par exemple, la valeur de recherche :

```
pe/lib/pretty/text
```

permet d'accéder aux entrées

```

perl/librairies/PrettyCurses/TextField.pm    rw-
perl/librairies/PrettyCurses/TextMemo.pm     rw-

```

Il est à noter que la différence minuscules / majuscules n'est pas prise en compte. Plus exactement, les recherches se font en tenant compte de la variable globale `$PC_order`.

La valeur de recherche

```
pe/lib/pretty/text                                rwx
```

n'aurait, quant à elle, reconnu aucune entrée.

Les champs de recherche peuvent contenir des jokers :

- ? Reconnaît n'importe quel caractère.
- \* Reconnaît n'importe quelle chaîne de caractères.

\$ Indique une fin de champ.

[...]

Propose une alternative. Par exemple [abc] reconnaît soit un a, soit un b, soit un c.

Pour le premier champ, il y a un dernier caractère spécial. La séquence « \*\* » reconnaît un nombre arbitraire de menus intermédiaires. Ainsi, la valeur de recherche :

**\*\*/\*.pm\$**

reconnaît toutes les entrées se terminant par .pm.

Finalement, il est possible d'introduire ces caractères sans leur donner le sens précis précédent en les protégeant par des « \ ».

## Paramètres

Ici ne sont détaillés que les paramètres spécifiques au widget Menu. Pour les paramètres généraux, se reporter à *PrettyCurses::Widgets(3)*. Les valeurs qui sont données au début de cette page sont celles qui sont passés par défaut si elles ne sont pas initialisées lors de l'appel de la fonction **new**.

### BORDER

Définit la taille de la bordure. S'il vaut 0, aucune bordure n'est dessinée et les modes **SCROLL** et **MODIFY** ne sont pas affichés dans la barre d'état.

### DEFIL\_BAR

Précise si une barre de défilement doit être affichée (cas **DEFIL\_BAR** => 1) ou si ce n'est pas nécessaire (cas **DEFIL\_BAR** => 0). Celle-ci n'est de toute façon pas affichée s'il n'y a pas de bordure.

**TREE** Voir le paragraphe suivant.

### ITEMS

Ce paramètre est un champ système ; il est donc tout à fait déconseillé d'y accéder directement, surtout en écriture. Il est créé automatiquement après une mise à jour des paramètres et est mis à jour dynamiquement lors de l'exécution du widget. Il contient la liste des entrées affichées sur l'écran auxquelles ont été ajoutées quelques pointeurs nécessaires à un déplacement efficace.

### LENGTHS

Longueur des champs d'une entrée. Attention, on rappelle qu'il s'agit d'une référence sur une liste comme cela a été expliqué précédemment.

### HEAD\_LINE

Numéro de la première entrée affichée.

### SELECTED\_ITEM

Numéro de l'entrée sélectionnée. Il vaut -1 si aucune entrée n'est sélectionnée.

### COLORS

Couleur par défaut d'une entrée. Là encore, il s'agit d'une référence sur une liste.

### MODE\_SCROLL

Active (cas **MODE\_SCROLL** => 1) ou désactive (cas **MODE\_SCROLL** => 0) le mode **SCROLL**. Voir plus loin pour plus d'informations.

### ENABLE\_SEARCH

Précise s'il faut afficher les champs de recherche (cas **ENABLE\_SEARCH** => 1) ou si ce n'est pas nécessaire (cas **ENABLE\_SEARCH** > 0).

### CURRENT\_FIELD

Numéro du champ de recherche ayant le focus. S'il vaut -1, aucun de ces champs n'a le focus.

## MAX\_DEEP

Nombre maximal de ramifications autorisées dans le premier champ de recherche. S'il vaut 1, le mode `SCROLL` n'est pas pris en considération.

## COLOR\_FIELD

Couleur utilisée pour les champs de recherche.

## MODE\_MODIFY

Active (cas `MODE_MODIFY => 1`) ou désactive (cas `MODE_MODIFY => 0`) le mode `MODIFY`. Il précise si les modifications faites dans le menu doivent se répercuter dans les champs de recherche et réciproquement. Voir plus loin pour plus d'informations.

## Description du paramètre TREE

Il s'agit d'une référence sur la liste des entrées proposées dans le menu. Chaque entrée, disons `$item`, est une référence sur une table de hash devant respecter le format suivant :

```
$item = {  CAPTION           =>      ,
          RETURN_VALUE      =>      ,
          COLORS            =>      ,
          SHOW_CHILDREN     =>      ,
          CHILDREN          =>      ,
          ARGV              =>      ,
          DYNAMIC           =>      };
```

## CAPTION

Intitulé de `$item`. Attention, on rappelle qu'il s'agit d'une référence sur une liste comme cela a été expliqué précédemment.

## RETURN\_VALUE

Valeur à laquelle est positionné le paramètre global `VALUE` lorsque l'entrée en question est sélectionnée.

## COLORS

Couleur de `$item`. Là encore, il s'agit d'une référence sur une liste. Si une couleur n'est pas définie, la couleur correspondante dans la table référencée par le paramètre `COLORS` est utilisée.

## SHOW\_CHILDREN

Précise s'il faut afficher les sous-menus disponibles dans cette entrée (cas `SHOW_CHILDREN => 1`) ou si ce n'est pas nécessaire (cas `SHOW_CHILDREN => 0`).

## CHILDREN

Deux cas sont possibles. 1) référence sur un objet de même type que `TREE` décrivant les entrées du sous-menu auquel on accède après avoir sélectionné `$item`. 2) référence sur un `CODE` dont la valeur de retour est comme décrit dans le cas précédent.

**ARGV** Référence sur la liste des arguments passée au code pointé par `CHILDREN`. Ce paramètre est ignoré dans le cas où `CHILDREN` ne contient pas une référence sur un `CODE`.

## DYNAMIC

Précise, dans le cas où `CHILDREN` contient une référence sur un `CODE`, si celui-ci doit être exécuté à chaque fois lorsqu'on désire accéder au sous-menu en question (cas `DYNAMIC => 1`) ou s'il n'est nécessaire d'exécuter le code que la première fois (cas `DYNAMIC => 0`).

## Méthodes

Se reporter à la page *PrettyCurses::Widgets(3)*.

## Touches interceptées lors de l'exécution

UP Sélectionne l'entrée précédente de même niveau.

DOWN Sélectionne l'entrée suivante de même niveau.

PGUP Sélectionne une entrée de même niveau suffisamment haute pour faire défiler l'écran

PGDOWN

Sélectionne une entrée de même niveau suffisamment basse pour faire défiler l'écran

LEFT Remonte au menu de niveau précédent. Efface le sous-menu fils si le mode `SCROLL` est actif.

SHIFT-LEFT

Remonte au menu de niveau précédent mais efface le sous-menu fils si le mode `SCROLL` est inactif.

RIGHT

Entre dans le sous-menu de l'entrée sélectionné s'il existe.

ENTER

Détaille le sous-menu de l'entrée sélectionné sans toutefois y entrer. La touche n'est pas interceptée s'il n'existe pas de tel sous-menu.

DELETE

Efface le sous-menu de l'entrée sélectionné au cas où il aurait été affiché.

+ Détaille récursivement l'entrée sélectionnée. Si aucune entrée n'est sélectionnée détaille récursivement tout l'arbre.

- Ordonne récursivement aux sous-menus de l'entrée sélectionnée de ne plus afficher leurs sous-menus. Si aucune entrée n'est sélectionnée, ordonne cela à tous les sous-menus de l'arbre.

CONTROL-UP

Sélectionne l'entrée précédente la plus proche reconnue par la valeur de recherche en cours. Cette touche n'est pas interceptée si le paramètre `ENABLE_SEARCH` est positionné à 0.

CONTROL-DOWN

Sélectionne l'entrée suivante la plus proche reconnue par la valeur de recherche en cours. Cette touche n'est pas interceptée si le paramètre `ENABLE_SEARCH` est positionné à 0.

CONTROL-END

Retire le focus de tous les champs de recherche. Cette touche n'est pas interceptée si le paramètre `ENABLE_SEARCH` est positionné à 0.

CONTROL-U

Efface la valeur de recherche actuelle. Le comportement global de `CONTROL-U` est donc le suivant. Si le champ en cours n'est pas vide, la valeur de ce champ est effacée. Si ce champ fait partie du premier champ de recherche et que celui-ci est vide, c'est ce premier champ de recherche tout entier qui est effacé s'il n'est pas vide. Si finalement cette touche est pressée sur un champ de recherche vide, c'est tous les champs de recherche qui sont simultanément effacés. Pour plus d'explications, consulter les pages *PrettyCurses::Widgets(3)*, *PrettyCurses::TextField(3)* et *PrettyCurses::MultiFields(3)*.

M ou ALT-M

Bascule le mode `MODIFY`.

S ou ALT-S

Bascule le mode `SCROLL`.

# PrettyCurses::directory

## NOM

PrettyCurses::directory

## SYNOPSIS

```
use PrettyCurses::directory;

$d = PrettyCurses::directory->new ({
    CAPTION           =>    [ "__NAME()" ],
    RETURN_VALUE     =>    [ "__FULLPATH()" ],
    COLORS            =>    [ ],
    SHOW_CHILDREN    =>    0,
    DYNAMIC           =>    0,

    CODE              =>    "__NAME()",
    re_CODE           =>    qr/^[^\./]/,
    MAX_DEEP_SYMLINK =>    5,

    HASH              =>    { }
});

$d->default ();

$d->getParam ($name);
$d->setParam ( $name1 => $value1, ... );
$d->setHash ( $name1 => $value1, ... );

$d->maketree ($dir);
```

## DESCRIPTION

Ce package permet de construire un objet pouvant servir de paramètre `TREE` d'un *PrettyCurses::Menu(3)* dénotant toute l'arborescence d'un certain répertoire.

Pour cela, il est d'abord nécessaire de définir un format via la fonction `new`. L'appel de la fonction `maketree` crée alors l'objet souhaité.

### Créer un nouveau format

Ceci se fait donc par l'intermédiaire de la fonction `new` dont la syntaxe a été détaillé précédemment. Les valeurs qui ont été assignées aux différents paramètres correspondent aux valeurs assignées par défaut. Si aucun argument n'est fourni lors de l'appel de la fonction `new`, ce sont ceux de la variable globale `$PC_directory_default` qui sont utilisés.

Mais détaillons tout de suite ce que sont ces paramètres.

`CAPTION`, `RETURN_VALUE`, `COLORS`, `SHOW_CHILDREN`, `DYNAMIC`

Correspondent aux paramètres analogues d'un objet de type `TREE` (voir *PrettyCurses::Menu(3)*). Ils peuvent contenir des codes spéciaux qui seront détaillés dans le paragraphe suivant.

CODE, re\_CODE

CODE est un champ de caractères quelconque pouvant inclure lui aussi les codes spéciaux du paragraphe suivant. Un fichier est ajouté à l'objet qui va être créé si et seulement si l'expression régulière re\_CODE reconnaît son CODE.

MAX\_DEEP\_SYMLINK

Profondeur maximale pour suivre les liens symboliques.

HASH Référence sur une table de hash servant aux codes spéciaux précédents.

## Description des codes spéciaux

Les codes spéciaux sont :

\_\_NAME()

Ce code est remplacé par le nom du fichier.

\_\_FULLPATH()

Ce code est remplacé par le chemin complet du fichier.

\_\_TYPE()

Ce code est remplacé par le type du fichier. Les types reconnus sont « file », « directory », « FIFO », « socket », « block » et « character ». Si le fichier est un lien symbolique le type est précédé de la chaîne de caractères « symlink/ ».

\_\_READABLE()

Ce code est remplacé par « r » si l'utilisateur peut lire le fichier en question, par « - » si ce n'est pas le cas.

\_\_WRITABLE()

Ce code est remplacé par « w » si l'utilisateur peut modifier le fichier en question, par « - » si ce n'est pas le cas.

\_\_EXECUTABLE()

Ce code est remplacé par « r » si l'utilisateur peut exécuter le fichier en question, par « - » si ce n'est pas le cas.

Si une chaîne de caractères, disons \$code, apparaît entre les parenthèses précédentes, les choses se passent de la façon suivante. C'est là qu'intervient le paramètre global HASH. Il est nécessaire de décrire sa structure. Il s'agit d'une référence sur une table de hash, disons \$hash ayant la structure suivante :

```
$hash = { $code1 => [ qr/$regexp1.1/ => $string1.1, ... ],
          $code2 => [ qr/$regexp2.1/ => $string2.1, ... ],
          ...
};
```

Dans le cas précédent, donc, on regarde si \$code apparaît comme clé dans la table de hash \$hash. Si ce n'est pas le cas, le comportement est le même que s'il n'y avait rien entre parenthèses. Si par contre, c'est le cas, on regarde dans l'ordre si la valeur qui devait être remplacée est reconnu par l'une des expressions régulières associées au code \$code. Si ce n'est pas le cas, le comportement est encore celui décrit précédemment. Si par contre c'est le cas, le code spécial est remplacée par la valeur correspondante.

Les \$string peuvent contenir des \$1, \$2, etc. Ils auront le comportement souhaité sauf s'ils sont protégés par des « \ ». Attention, les \$string doivent vraiment être des chaînes de caractères ; il n'est par exemple pas question de définir ainsi une couleur par sa table de hash.

Finalement, il est possible de protéger ces codes spéciaux par des « \ ».

## Méthodes

`$d->default ()`;

Réinitialise les paramètres à ceux de la variable globale `$PC_directory_default`.

`$d->getParam ($name)`;

Renvoie la valeur du paramètre `$name`.

`$d->setParam ( $name1 => $value1, ... )`;

Affecte la valeur `$value1` au paramètre `$name1` et ainsi de suite.

`$d->setHash ( $code1 => [ qr/$regex1.1/ => $string1.1, ... ], ... )`;

Affecte à la clé `$code1` de la table de hash référencée par le paramètre `HASH` la valeur correspondante, et ainsi de suite.

`$d->maketree ($dir)`;

Renvoie un objet de type `TREE` qui décrit l'arborescence du répertoire `$dir`.

# PrettyCurses::CheckBox

## NOM

PrettyCurses::CheckBox

## SYNOPSIS

```
use PrettyCurses::CheckBox;

$w = PrettyCurses::CheckBox->new ({
    X           => 0,
    Y           => 0,
    VALUE       => 0,
    KEYS        => { },
    READONLY    => 0,
    UNDO        => 1,
    COLOR       => 'fields',
    DRAW_AFTER_EXECUTE => 1
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );
$w->getValue ();
$w->setValue ($value);

$w->draw ($PC_win, $focus);
$w->execute ($PC_win);

$w->undo ($PC_win, $focus);
$w->redo ($PC_win, $focus);
$w->save_position ($name);
$w->remove_position ($name);
$w->addinhistory ($name);
$w->clearhistory ();
```

## DESCRIPTION

Ce package implémente un widget CheckBox (case à cocher). Seulement la case est gérée par ce package. Pour rajouter une légende, se reporter à *PrettyCurses::Caption(3)* ou encore à *PrettyCurses::Form(3)*.

## Paramètres

Pour les paramètres généraux, se reporter à la page *PrettyCurses::Widgets(3)*. Les valeurs qui sont données ci-dessus sont celles qui sont passés par défaut si elles ne sont pas initialisées lors de l'appel de la fonction `new`.

Le paramètre `VALUE` est positionné à 1 lorsque la case est cochée et à 0 si elle ne l'est pas.

## Méthodes

Se reporter à la page *PrettyCurses::Widgets(3)*.

## **Touches interceptées lors de l'exécution**

### **SPACE**

Coche ou décoche la case.

### **CONTROL-PGUP**

Remonte d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà au début de l'historique. Cette touche est de toute façon ignorée si le paramètre **UNDO** est positionné à 0.

### **CONTROL-PGDOWN**

Descend d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà à la fin de l'historique. Cette touche est de toute façon ignorée si le paramètre **UNDO** est positionné à 0.

# PrettyCurses::Button

## NOM

PrettyCurses::Button

## SYNOPSIS

```
use PrettyCurses::Button;

$w = PrettyCurses::Button->new ({
    X           => 0,
    Y           => 0,
    HEIGHT_X    => length (CAPTION) + 4,
    HEIGHT_Y    => 1,
    CAPTION     => 'OK',
    KEYS        => { },
    COLOR       => 'default',
    DRAW_AFTER_EXECUTE => 1
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );

$w->draw ($PC_win, $focus);
$w->execute ($PC_win);
```

## DESCRIPTION

Ce package implémente un widget Button (bouton).

## Paramètres

Pour les paramètres généraux, se reporter à la page *PrettyCurses::Widgets(3)*. Les valeurs qui sont données ci-dessus sont celles qui sont passés par défaut si elles ne sont pas initialisées lors de l'appel de la fonction `new`.

Le paramètre `CAPTION` correspond au texte noté sur le bouton.

## Méthodes

Se reporter à la page *PrettyCurses::Widgets(3)*.

## Touches interceptées lors de l'exécution

### ENTER

Arrête l'exécution du widget en renvoie le code « `/BUTTONPRESS` » suivi de la valeur du paramètre `CAPTION`. Si le paramètre `KEYS` précisait qu'il fallait intercepter la touche `ENTER`, cette interception est prioritaire.

# PrettyCurses::Form

## NOM

PrettyCurses::Form

## SYNOPSIS

```
use PrettyCurses::Form;

$w = PrettyCurses::Form->new ({
    X           => 0,
    Y           => 0,
    HEIGHT_X    => 10,
    HEIGHT_Y    => 10,
    KEYS        => { },
    READONLY    => 0,
    UNDO        => 0,
    CORRECT     => 1,
    COLOR       => 'default',
    CLEAR       => 1,
    BORDER      => 1,
    DEFIL_BAR   => 1,
    DRAW_BEFORE_EXECUTE => 0,
    DRAW_AFTER_EXECUTE  => 0,

    HEAD_LINE   => 0,
    CURRENT     => '__first',

    EMPTYLINES_BEFORE => 0,
    X_FIELDS    => ,
    HEIGHT_X_FIELDS => ,
    HEIGHT_Y_FIELDS => 1,
    VALUE       => ,
    EXECUTE     => 1,
    UNDO_FIELDS => ,
    CORRECT_FIELDS => ,
    COLOR_FIELDS => ,
    COLOR_ERR   => ,
    DAE_FIELDS  => ,
    CAPTION_Xrel => 0,
    CAPTION_Yrel => 0,
    CAPTION_COLOR => 'default',

    RECURSIVE  => 2
});

$w->getParam ($name);
$w->setParam ( $name1 => $value1, ... );
$w->getField ($field_name);
$w->setField ($field_name, { $name1 => $value1, ... });
```

```

$w->getWidget ($field_name);
$w->getFields ();
$w->getValues ();

$w->append ($field1, ...);
$w->add_before ($field_name, $field1, ...);
$w->add_after ($field_name, $field1, ...);
$w->delete ($field_name);

$w->draw ($PC_win, $focus);
$w->execute ($PC_win);

$w->correct ($PC_win, $focus);

$w->undo ($PC_win, $focus);
$w->redo ($PC_win, $focus);
$w->save_position ($name);
$w->remove_position ($name);
$w->addinhistory ($name);
$w->clearhistory ();

```

## DESCRIPTION

Ce package implémente un widget Form. Il s'agit d'un formulaire pouvant contenir nombre d'autres widgets qui sont donc gérés ici.

## Paramètres

Ici ne sont détaillés que les paramètres spécifiques au widget Menu. Pour les descriptions générales, se reporter à la page *PrettyCurses::Widgets(3)*. Les valeurs qui sont données ci-dessus sont celles qui sont passés par défaut si elles ne sont pas initialisées lors de l'appel de la fonction `new`.

### CLEAR

Précise s'il faut effacer l'ancien formulaire dessiné avant d'en dessiner un nouveau via la commande `draw`. Il peut être utile de positionner ce paramètre à 0 si l'on fait beaucoup d'appels à la fonction `draw` afin que l'écran ne *clignote* pas trop.

### BORDER

Précise la taille de la bordure. Si ce paramètre est positionné à 0, aucune bordure n'est dessinée.

### DEFIL\_BAR

Précise s'il faut dessiner une barre de défilement (cas `DEFIL_BAR => 1`) ou si ce n'est pas nécessaire (cas `DEFIL_BAR => 0`). Si aucune bordure n'est dessinée, la barre de défilement est de toute façon ignorée.

### DRAW\_BEFORE\_EXECUTE

Précise s'il faut redessiner le formulaire avant l'exécution (cas `DRAW_BEFORE_EXECUTE => 1`) ou si ce n'est pas nécessaire (cas `DRAW_BEFORE_EXECUTE => 0`).

### HEAD\_LINE

Numéro de la première ligne affichée à l'écran.

### CURRENT

Nom du champ courant.

## RECURSIVE

Stipule si les paramètres globaux doivent être passés automatiquement aux divers champs. Si **RECURSIVE** est positionné à 0, la fonction **setField** (détaillée plus loin) n'est jamais appelée automatiquement. S'il est positionné à 1, la fonction **setField** est appelée automatiquement sur tous les champs nouvellement créés. S'il est positionné à 2, la fonction **setField** est de plus appelée après sur tous les champs après chaque appel à la fonction **setParam**. S'il est positionné à une valeur négative via la commande **setParam**, aucun appel à **setField** n'est fait lors de ce positionnement, mais le signe est supprimé de sorte que les appels suivants pourront tenir compte de la valeur affectée.

Les autres paramètres ne se rapportent pas à proprement parler au formulaire. Ils sont simplement passés par défaut aux divers champs que le formulaire va contenir.

## Format des champs du formulaire

Les champs du formulaires sont stockées dans une liste doublement chaînée (implémentée au moins d'une table de hash). Cette liste contient deux éléments particuliers qui sont « **\_\_first** » et « **\_\_last** » qui sont situés respectivement avant le premier champ et après le dernier. Les autres éléments de cette liste sont des tables de hash qui doivent obéir au format suivant :

```
$field = {  NAME           =>      ,
           EMPTYLINES_BEFORE =>      ,
           WIDGET           =>      ,
           X_FIELDS         =>      ,
           HEIGHT_X_FIELDS  =>      ,
           HEIGHT_Y_FIELDS  =>      ,
           VALUE            =>      ,
           EXECUTE          =>      ,
           READONLY         =>      ,
           UNDO_FIELDS      =>      ,
           CORRECT_FIELDS   =>      ,
           COLOR_FIELDS     =>      ,
           COLOR_ERR        =>      ,
           DAE_FIELDS       =>      ,
           CAPTION          =>      ,
           CAPTION_Xrel     =>      ,
           CAPTION_Yrel     =>      ,
           CAPTION_COLOR    =>      };
```

Les valeurs par défaut données à ces paramètres sont celles des paramètres globaux correspondant s'ils existent.

**NAME** Nom du champ. Si aucun nom n'est passé, un nom sera attribué automatiquement. Il peut être parfois utile de ne pas s'encombrer de noms mais ce n'est pas forcément une bonne idée.

### EMPTYLINES\_BEFORE

Nombre de lignes à sauter avant d'afficher le champ en question. Ce nombre peut très bien être négatif.

### WIDGET

Widget implémentant le champ en question. Il n'est demandé au dit widget que de connaître les paramètres et les méthodes présentés dans *PrettyCurses::Widgets*.

### X\_FIELDS

Position horizontale du champ. Ce nombre peut être négatif auquel cas la position est comptée à partir du bord droit du formulaire.

#### HEIGHT\_X\_FIELDS

Taille horizontale du champ. Ce nombre peut-être négatif auquel cas le champ occupera toute la largeur du formulaire sauf le nombre de caractères précisé.

#### HEIGHT\_Y\_FIELDS

Taille verticale du champ. Ce nombre peut-être négatif auquel cas le champ occupera toute la hauteur du formulaire sauf le nombre de ligne précisé.

#### VALUE

Valeur du champ. Cette valeur est passée au widget via la commande `setValue`. Ainsi, si par exemple le widget est un `PrettyCurses::TextField(3)`, elle devra être renseignée à " `Bonjour` " et non à [ [ "Bonjour" ] ]. Voir `PrettyCurses::TextField(3)` pour plus d'informations.

#### EXECUTE

Précise si ce widget doit être exécuté (cas `EXECUTE => 1`) ou si on doit simplement y passer dessus (cas `EXECUTE => 0`).

#### READONLY

Précise si le champ doit être en lecture seule (cas `READONLY => 1`) ou si ce n'est pas le cas (cas `READONLY => 0`). Si le paramètre global `READONLY` est positionné à 1, le champ sera de toute façon en lecture seule.

#### UNDO\_FIELDS

Précise si le champ doit retenir son historique et gérer les combinaisons de touches `CONTROL-PGUP` et `CONTROL-PGDOWN` (cas `UNDO_FIELDS => 1`) ou si ce n'est pas nécessaire (cas `UNDO_FIELDS => 0`). Si le paramètre global `UNDO` est positionné à 1, ce paramètre sera affecté de la valeur 0 quoi qu'il arrive. Il est conseillé, pour des question de légèreté, de laisser les widgets gérer leur propre historique et de désactiver cette option pour le formulaire.

#### CORRECT\_FIELDS

Précise si le champ doit intercepter la combinaison de touches `CONTROL-T` afin de faire les corrections automatiques (cas `CORRECT_FIELDS => 1`) ou si ce n'est pas nécessaire (cas `CORRECT_FIELDS => 0`).

#### COLOR\_FIELDS

Couleur du champ.

#### COLOR\_ERR

Couleur que le champ doit utiliser pour les erreurs. Voir `PrettyCurses::TextField(3)`.

#### DAE\_FIELDS

Les initiales « `DAE` » signifient « `DRAW_AFTER_EXECUTE` ». Ce paramètre précise si le champ doit être redessiné après avoir été exécuté (cas `DAE_FIELDS => 1`) ou si ce n'est pas nécessaire (cas `DAE_FIELDS => 0`). Le champ courant n'est jamais redessiné si un événement `ON_PRESS` ou `ON_MODIFY` est intercepté.

#### CAPTION

Texte de légende servant à introduire le champ.

#### CAPTION\_Xrel et CAPTION\_Yrel

Position du texte précédent comptée relativement par rapport à la position du champ.

#### CAPTION\_COLOR

Couleur qu'il faut utiliser pour écrire le texte précédent.

## Méthodes

Ici ne sont détaillés que les méthodes spécifiques au widget `Menu`. Pour les descriptions générales, se reporter à la page `PrettyCurses::Widgets(3)`.

## Gestion des paramètres

`$w->getField ($field_name, $name);`

Renvoie la valeur du paramètre `$name` du champ nommé `$field_name`. Renvoie `undef` si le champ `$field_name` n'existe pas.

`$w->setField ($field_name, { $name1 => $value1, ... });`

Met à jour la table de hash du champ `$field_name`.

`$w->getWidget ($field_name);`

Renvoie le widget du champ `$field_name`.

## Gestion des champs

`$w->getFields ();`

Renvoie la liste des champs présents sur le formulaire dans l'ordre.

`$w->getValues ();`

Renvoie une liste qui associe à chaque nom de champ sa valeur, cette valeur étant récupérée via la commande `getValue`. La format de la liste renvoyée est (`$name1, $value1, $name2, $value2, ...`).

`$w->append ($field1, ...);`

Ajoute les champs `$field1, $field2, etc.` à la fin. Les arguments `$field` doivent être des tables de hash respectant le format décrit au chapitre précédent.

`$w->add_before ($field_name, $field1, ...);`

Ajoute les champs `$field1, $field2, etc.` juste avant le champ nommé `$field_name`. Ne fait rien si le champ `$field_name` n'existe pas. Les arguments `$field` doivent être des tables de hash respectant le format décrit au chapitre précédent.

`$w->add_after ($field_name, $field1, ...);`

Ajoute les champs `$field1, $field2, etc.` juste après le champ nommé `$field_name`. Ne fait rien si le champ `$field_name` n'existe pas. Les arguments `$field` doivent être des tables de hash respectant le format décrit au chapitre précédent.

`$w->delete ($field_name);`

Supprime le champ nommé `$field_name` du formulaire. Ne teste pas si malencontreusement le champ courant donné par le paramètre global `CURRENT` était précisément celui-ci.

## Touches interceptées lors de l'exécution

Bien entendu, `PrettyCurses::Menu(3)` fait appel à chacun des widgets proposés. Il est donc très fréquent que le widget appelé intercepte la touche sans que `PrettyCurses::Menu(3)` soit au courant. Normalement, tout est fait correctement pour que le comportement attendu soit celui qui se passe réellement.

**LEFT** Se déplace à la fin du champ précédent. Si le focus était déjà donné au premier champ, la touche n'est pas interceptée.

**UP** Se déplace au début du champ précédent. Si le focus était déjà donné au premier champ, la touche n'est pas interceptée.

**RIGHT, DOWN, TAB, ENTER**

Se déplace au début du champ suivant. Si le focus était déjà sur le dernier champ, la touche n'est pas interceptée.

**PGUP** Se déplace au début d'un champ précédent suffisamment loin pour faire défiler l'écran.

**PGDOWN**

Se déplace au début d'un champ suivant suffisamment loin pour faire défiler l'écran.

#### **DOUBLE CONTROL-T**

Fait les corrections automatiques sur tous les champs du formulaire pour lesquels le paramètre `CORRECT_FIELDS` est positionné à 1. La touche **DOUBLE CONTROL-T** s'obtient en double-appuyant (comme double-cliquant mais au clavier) sur la combinaison de touches **CONTROL-T**. La touche est de toute façon ignorée si le paramètre `READONLY` est positionné à 1.

#### **CONTROL-PGUP**

Remonte d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà au début de l'historique. Cette touche est de toute façon ignorée si le paramètre `UNDO` est positionné à 0 ou si le paramètre `READONLY` est positionné à 1.

#### **CONTROL-PGDOWN**

Descend d'un cran dans l'historique. La touche n'est pas interceptée si on était déjà à la fin de l'historique. Cette touche est de toute façon ignorée si le paramètre `UNDO` est positionné à 0 ou si le paramètre `READONLY` est positionné à 1.