perllocale

Table des matières

1 NAME/NOM

perllocale - Gestion des "locale" en Perl (internationalisation et localisation)

2 DESCRIPTION

Perl supporte pour les données, des notions spécifiques à la langue telles que "Est-ce une lettre", "quel est l'équivalent en majuscule de cette lettre" et "laquelle de ces lettres est la première". Ce sont des sujets très importants spécialement pour les langues autres que l'anglais – mais aussi pour l'anglais : il serait naïf d'imaginer que `A-Za-z` décrit toutes les "lettres" nécessaires à l'écriture de l'anglais. Perl sait aussi qu'un autre caractère que '.' peut être utilisé comme séparateur décimal et que la représentation d'une date est liée à la langue. La tâche consistant à rendre une application capable de gérer les préférences de l'utilisateur dans ce domaine s'appelle **l'internationalisation** (parfois abrégé en **i18n**). Spécifier à une telle application l'ensemble de ces préférences pour une langue particulière s'appelle la **localisation** (**l10n**).

Perl peut comprendre les données spécifiques à une langue via la méthode standard (ISO C, XPG4, POSIX 1.c) appelée système de "locale". Le système de "locale" est contrôlé par une application en utilisant un pragma (une directive), un appel de fonction et plusieurs variables d'environnement.

Note : cette fonctionnalité est nouvelle dans Perl 5.004 et ne doit pas s'appliquer à moins qu'une application le demande explicitement – voir Compatibilité (§8.1). La seule exception est write() qui maintenant utilise **toujours** le "locale" courant – voir NOTES (§8).

3 PRÉ-REQUIS POUR L'UTILISATION DES "LOCALE"

Pour qu'une application Perl comprenne et présente vos données correctement en fonction du "locale" de votre choix, **toutes** les conditions suivantes doivent être remplies :

– **Votre système d'exploitation doit supporter le système de "locale"**. Si c'est le cas, vous devriez trouver la fonction setlocale() dans la documentation sa bibliothèque C.
– **Les définitions des "locale" que vous voulez utiliser doivent être installées**. Vous ou votre administrateur système devez être sûr que c'est bien le cas. Les "locale" disponibles, l'endroit où ils sont stockés et la manière dont ils sont installés varient d'un système à l'autre. Certains systèmes ne fournissent que très peu de "locale" installés une bonne fois pour toutes et il n'est pas possible d'en ajouter. D'autres autorisent l'ajout de "locale" en conserve fournis par le fournisseur du système. D'autres encore vous autorisent à créer et à définir vos propres "locale". Lisez la documentation de votre système pour d'autres informations.
– **Perl doit savoir que le système de "locale" est supporté**. Si c'est le cas, `perl -V:d_setlocale` doit dire que la valeur de `d_setlocale` est `define`.

Si vous voulez qu'une application Perl traite et présente les données en fonction d'un "locale" particulier, le code de l'application devrait inclure la directive `use locale` (voir La directive locale) là où c'est approprié et **au moins** l'une des conditions suivantes devraient être remplie :

– **les variables d'environnement (voir ENVIRONNEMENT (§7)) qui déterminent le "locale" doivent être correctement positionnées** au moment du lancement de l'application soit par vous-même soit par la configuration de votre compte système.
– **L'application doit positionné son propre "locale"** en utilisant la méthode décrite dans La fonction setlocale.

4 UTILISATION DES "LOCALE"

4.1 La directive locale

Par défaut, Perl ignore le "locale" courant. La directive `use locale` demande à Perl de prendre en compte le "locale" courant pour certaines opérations :

– **Les opérateurs de comparaisons** (`lt`, `le`, `cmp`, `ge` et `gt`) et les fonctions POSIX de tri strcoll() et strxfrm() utilisent `LC_COLLATE`. sort() est aussi affecté si il est utilisé sans une fonction de comparaison explicite puisqu'il utilise `cmp` par défaut.
Note: `eq` et `ne` ne sont pas affectés par le "locale" : ils comparent toujours leurs deux opérandes octet par octet. De plus, si `cmp` trouve que ses deux opérandes sont égaux selon l'ordre spécifié par le "locale" courant, il lance aussi une comparaison octet par octet et retourne *0* (égal) uniquement si les deux opérandes sont identiques bit à bit. Si vous voulez vraiment savoir si deux chaînes - que `eq` et `cmp` considèrent comme différentes – sont égales selon l'ordre de tri du "locale", lisez Catégorie LC_COLLATE : tri.
– **Les expressions rationnelles et les fonctions de modification de la casse** (uc(), lc(), ucfirst() et lcfirst()) utilisent `LC_CTYPE`.
– **Les fonctions de formatage** (printf(), sprintf() et write()) utilisent `LC_NUMERIC`.
– **La fonction POSIX de formatage de dates** (strftime()) utilise `LC_TIME`.

`LC_COLLATE`, `LC_CTYPE` et autres sont présentés plus complètement dans CATÉGORIES DE "LOCALE".

Le comportement par défaut est rétabli par la directive `no locale` ou lorsqu'on sort du bloc englobant la directive `use locale`.

Une chaîne résultat de n'importe quelle opération qui utilise les informations du "locale" est souillée (tainted) puisqu'il vaut mieux ne pas faire confiance au "locale" courant. Voir SÉCURITÉ (§6).

4.2 La fonction setlocale

Vous pouvez changer de "locale" aussi souvent que nécessaire lors de l'exécution grâce à la fonction POSIX::setlocale() :

```
# Cette fonctionnalité n'existait pas avant Perl 5.004
require 5.004;

# Import des outils de manipulation du "locale"
# depuis le module POSIX.
# Cette exemple utilise : setlocale -- l'appel de fonctio
#                         LC_CTYPE -- expliqué plus bas
use POSIX qw(locale_h);

# demande et sauvegarde le "locale" initial
$old_locale = setlocale(LC_CTYPE);

setlocale(LC_CTYPE, "fr_CA.ISO8859-1");
# LC_CTYPE est maintenant dans le
# "locale" "French, Canada, codeset ISO 8859-1"

setlocale(LC_CTYPE, "");
# LC_CTYPE revient à sa valeur par défaut définie par
# les variables d'environnement LC_ALL/LC_CTYPE/LANG
# Voir plus bas pour la documentation

# restaure le "locale" initial
setlocale(LC_CTYPE, $old_locale);
```

Le premier argument de setlocale() donne la **catégorie**, le seconde le **locale**. La catégorie indique à quel aspect du traitement des données vous voulez appliquer les règles spécifiques au "locale". Les noms des catégories sont présentés dans CATÉGORIES DE "LOCALE" et ENVIRONNEMENT (§7). Le "locale" est le nom d'une collection d'information correspondant à une combinaison particulière de langues, de pays ou territoires et de codage. Petit détail sur le nommage des "locale" : tous les systèmes ne nomment pas les "locale" comme dans les exemples donnés.

Si le second argument n'est pas fourni et que la catégorie est autre chose que LC_ALL, la fonction retourne une chaîne indiquant le "locale" courant pour cette catégorie. Vous pouvez utiliser cette valeur comme second argument d'un appel à setlocale().

Si le le second argument n'est pas fourni et que la catégorie est LC_ALL, le résultat est dépendant de l'implémentation. Cela peut être une chaîne concaténant tous les noms des "locale" (avec un séparateur dépendant de l'implémentation) ou un seul nom de "locale". Consultez *setlocale*(3) pour plus de détails.

Si un second argument est fourni et si il correspond à un "locale" valide, le "locale" pour cette catégorie est positionné à cette valeur et la fonction retourne la valeur du "locale" devenu le "locale" courant. Vous pouvez alors l'utiliser dans un autre appel à setlocale(). (Quelques implémentations retourne une valeur différente de la valeur que vous avez fournie comme second argument – pensez-y comme si vos valeurs étaient des alias).

Comme dans les exemples présentés, si le second argument est une chaîne vide, le "locale" de la catégorie revient à sa valeur par défaut spécifiée par les variables d'environnement correspondantes. Généralement, cela vous ramènera aux valeurs par défaut lorsque Perl a démarré : les changements de l'environnement faits par votre application après le démarrage peuvent ou non être pris en compte. Cela dépend de la bibliothèque C de votre système.

Si le second argument ne correspond à aucun "locale" valide, le "locale" pour cette catégorie n'est pas modifié et la fonction retourne *undef*.

Pour de plus amples informations concernant les catégories, consultez *setlocale*(3).

4.3 Trouver les "locale"

Pour connaître les "locale" disponibles sur votre système, consultez *setlocale*(3) pour savoir où réside la liste des "locale" disponibles (cherchez dans la section *SEE ALSO* ou *VOIR AUSSI*). Si cela échoue, essayez les lignes de commandes suivantes :

```
locale -a

nlsinfo

ls /usr/lib/nls/loc

ls /usr/lib/locale

ls /usr/lib/nls

ls /usr/share/locale
```

et regardez si elles listent quelque chose qui ressemble à :

```
en_US.ISO8859-1     de_DE.ISO8859-1     ru_RU.ISO8859-5
en_US.iso88591      de_DE.iso88591      ru_RU.iso88595
en_US               de_DE               ru_RU
en                  de                  ru
english             german              russian
english.iso88591    german.iso88591     russian.iso88595
english.roman8                          russian.koi8r
```

Malheureusement, bien que l'interface d'appel à setlocale() ait été standardisée, les noms des "locale" et les répertoires où la configuration réside ne le sont pas. La forme basique du nom est *langue_territoire***.***codage* mais les dernières parties après la langue ne sont pas toujours présentes. La *langue* et le *pays* sont habituellement sous la forme proposée par les normes **ISO 3166** et **ISO 639**, les noms de pays et de langues abrégés sur deux lettres. La partie *codage* mentionne parfois un jeu de caractères **ISO 8859**. Par exemple, `ISO 8859-1` est parfois appelé « jeu de caractères de l'Europe de l'Ouest » et peut être utilisé pour encoder la plupart des langues de l'Europe de l'Ouest. Même là, il y a plusieurs façons d'écrire le nom de ce standard. Lamentable.

Deux "locale" méritent une mention particulière : "C" et "POSIX". Pour l'instant, ce sont effectivement les mêmes "locale" : la seule différence est que l'un est défini par le standard C et l'autre par le standard POSIX. Ils définissent le **locale par défaut** que tous les programmes utilisent au démarrage en l'absence d'informations de "locale" dans leur environnement. Leur langue est l'anglais (américain) et leur jeu de caractères est l'ASCII.

Note : tous les systèmes n'ont pas le "locale" POSIX (tous les systèmes ne sont pas conformes POSIX). Donc utilisez "C" à chaque fois que vous avez besoin de spécifier le "locale" par défaut.

4.4 PROBLÈMES DE "LOCALE"

Vous pouvez obtenir le message d'avertissement suivant au démarrage de Perl :

```
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
        LC_ALL = "En_US",
        LANG = (unset)
    are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").
```

Cela signifie que vos réglages de "locale" ont LC_ALL positionné à "En_US" et que LANG existe mais n'a pas de valeur. Perl essaye de vous croire mais il ne le peut pas. À la place, Perl y renonce et retombe sur le "locale" "C", la "locale" par défaut qui est supposé fonctionner en toutes circonstances. Cela signifie que vos réglages de "locale" sont mauvais. Soit ils mentionnent des "locale" dont votre système n'a jamais entendu parler, soit l'installation des "locale" est mal faite sur votre système (par exemple, certains fichiers systèmes sont défectueux ou manquants). Il y a des solutions rapides et temporaires à ces problèmes mais aussi des solutions plus minutieuses et définitive.

4.5 Résolution temporaire des problèmes de "locale"

Les deux solutions les plus rapides sont soit de rendre Perl silencieux sur d'éventuelles inconsistances des "locale" soit de faire tourner Perl avec le "locale" par défaut ("C").

Les plaintes de Perl au sujet des problèmes de "locale" peuvent être supprimées en positionnant la variable d'environnement PERL_BADLANG à la valeur zéro (par exemple "0"). Cette méthode ne fait que cacher le problème : vous demandez à Perl de se taire même si il voit que quelque chose ne va pas. Ne soyez donc pas surpris, plus tard, lorsque vous constaterez des dysfonctionnements de choses dépendant des "locale".

Perl peut tourner avec le "locale" "C" par défaut en positionnant la variable d'environnement LC_ALL à "C". Cette méthode est peut-être un peu plus civilisée que l'approche par PERL_BADLANG mais la valeur de LC_ALL (ou de tout autre variable liée au "locale") peut affecter aussi d'autres programmes en plus de Perl. En particulier vos programmes externes lancés depuis Perl verront ces changements. Si vous rendez ces nouveaux réglages permanents, tous les programmes que vous ferez tourner les verront. Voir ENVIRONNEMENT (§7) pour une liste complète des variables d'environnement pertinentes et UTILISATION DES "LOCALE" pour leurs effets sur Perl. Les effets sur les autres programmes sont aisément déduits. Par exemple, la variables LC_COLLATE peut très bien influencer votre programme **sort** (ou plutôt le programme qui trient des 'enregistrements' dans l'ordre alphabétique, quel que soit son nom sur votre système).

Vous pouvez tester des changements temporaires sur ces variables puis, lorsque les nouveaux réglages semblent améliorer les choses, placer ces réglages dans l'un des fichiers de démarrage de votre shell. Consultez votre documentation locale pour les détails. Pour les shells de type Bourne-shell (**sh**, **ksh**, **bash**, **zsh**), cela donne :

```
LC_ALL=en_US.ISO8859-1
export LC_ALL
```

Cela suppose que vous avez vu le "locale" "en_US.ISO8859-1" en utilisant l'une des commandes présentées plus haut. Nous avons décidé d'essayer celui-là à la place du "locale" fautif "En_US". En shell type C-shell (**csh**, **tcsh**), cela donne :

```
setenv LC_ALL en_US.ISO8859-1
```

Si vous ne connaissez pas le shell que vous utilisez, consultez votre aide en ligne.

4.6 Résolution permanente des problèmes de "locale"

La méthode plus langue mais définitive de résoudre les problèmes de "locale" est de corriger vous-même la mauvaise installation ou configuration. La correction de la mauvaise (ou inexistante) installation peut nécessiter l'aide d'un administrateur système complaisant.

Tout d'abord, voyez plus haut ce qui concerne Trouver les "locale". Cela vous explique comment savoir les "locale" réellement supportés - et, plus important, installés – sur votre système. Dans notre message d'erreur d'exemple, les variables d'environnement affectant les "locale" sont listées dans l'ordre décroissant d'importance (la variables non positionnées ne comptent pas). Par conséquent, avoir LC_ALL fixé à "En_US" doit être un mauvais choix tel que se présente notre message d'erreur. Essayez donc de corriger les réglages de "locale" listés en premier.

En second, si en utilisant les commandes suggérées, vous voyez **exactement** quelque chose comme "En_US" sans les guillemets alors cela devrait fonctionner puisque vous utilisez alors un "locale" qui est censé être disponible sur votre système. Dans ce cas, voyez Résolution permanente des problèmes de configuration système des "locale".

4.7 Résolution permanente des problèmes de configuration système des "locale"

C'est lorsque vous voyez quelque chose comme :

```
perl: warning: Please check that your locale settings:
        LC_ALL = "En_US",
        LANG = (unset)
    are supported and installed on your system.
```

mais que vous ne trouvez pas "En_US" dans les listes produites par les commandes mentionnées plus haut. Vous pouvez voir des choses comme "en_US.ISO8859-1" mais ce n'est pas la même chose. Dans ce cas, essayez le "locale" de la liste qui ressemble le plus à celui que vous aviez précédemment. Les règles pour trouver les noms des "locale" sont un peu vagues car il y a un manque de standardisation dans ce domaine. Voir Trouver les "locale" pour les règles générales.

4.8 Configuration système des "locale"

Contactez un administrateur système (de préférence le vôtre), rapportez lui exactement le message que vous avez obtenu et demandez lui de lire la documentation que vous lisez présentement. Il devrait être en mesure de trouver ce qui ne va pas dans le configuration des "locale" de votre système. Trouver les "locale" est malheureusement un peu vague à propos des commandes et des emplacements car tout cela n'est pas standardisé.

4.9 La fonction localeconv

La fonction POSIX::localeconv() vous permet d'obtenir toutes les informations particulières liées au formatage des valeurs numériques selon le "locale" spécifié par `LC_NUMERIC` et `LC_MONETARY`. (Si vous voulez juste le nom du "locale" courant pour une catégorie particulière, utilisez POSIX::setlocale() avec un seul argument – voir La fonction setlocale.)

```
use POSIX qw(locale_h);

# Obtention d'une référence vers une table de hachage
# des information dépendant du "locale"
$locale_values = localeconv();

# Affichage de la liste triée des valeurs
for (sort keys %$locale_values) {
    printf "%-20s = %s\n", $_, $locale_values->{$_}
}
```

localeconv() ne prend aucun argument et retourne **une référence** vers une table de hachage. Les clés de cette table sont des noms des variables de formatage telle que `decimal_point` et `thousands_sep`. Les valeurs sont celles associées à cette clé. Voir localeconv in *POSIX* pour un exemple plus long listant les catégories qu'une implémentation devrait fournir; certaines en fournissent plus, d'autres moins. Vous n'avez pas besoin d'un `use locale` explicite puisque localeconv() respecte toujours le "locale" courant.

Voici un programme d'exemple simple qui réécrit les paramètres de sa ligne de commande comme des entiers correctement formatés selon le "locale" courant :

```
# Voir les commentaires des exemples précédents
require 5.004;
use POSIX qw(locale_h);

# Obtention de quelques=uns des paramètres
# de formattage numérique
my ($thousands_sep, $grouping) =
     @{localeconv()}{'thousands_sep', 'grouping'};

# Valeur par défaut
$thousands_sep = ',' unless $thousands_sep;
```

```
# grouping et mon_grouping sont sous la forme d'un liste compacte de
# petits entiers (caractères) indiquant le regroupement (thousand_seps
# et mon_thousand_seps spécifiant les séparateurs de groupes) des
# chiffres dans les nombres et la valeurs monétaires. La signification
# de ces entiers est : 255 pour indiquer qu'il ne faut plus regrouper,
# 0 pour indiquer de répéter le groupe précédent, 1-254 pour indiquer
# la taille du groupe courant. Les regroupements ont lieu de la droite
# vers la gauche. Dans ce qui suit, nous trichons un peu en
# n'utilisant que la première valeur de grouping.
if ($grouping) {
    @grouping = unpack("C*", $grouping);
} else {
    @grouping = (3);
}

# Présentation des paramètres de la ligne de commandes
# selon le "locale" courant
for (@ARGV) {
    $_ = int;    # Suppresions de la partie non entière
    1 while
    s/(\d)(\d{$grouping[0]}($|$thousands_sep))/$1$thousands_sep$2/;
    print "$_";
}
print "\n";
```

5 CATÉGORIES DE "LOCALE"

Les sous-sections suivantes décrivent les catégories de base de "locale". En plus, certaines combinaisons de catégories permettent la manipulation de plusieurs catégories à la fois. Voir ENVIRONNEMENT (§7).

5.1 Catégorie LC_COLLATE: tri

Dans la portée de `use locale`, Perl regarde la variable d'environnement `LC_COLLATE` pour déterminer la notion de collation (ordre) des caractères. Par exemple, 'b' suit 'a' dans l'alphabet Latin mais où se situent 'á' et 'å' ? Et bien que 'color' suive 'chocolate' en Anglais, qu'en est-il en Espagnol ?

Les collations suivantes sont toutes sensées et vous pouvez toutes les rencontrer si vous utilisez "use locale".

```
A B C D E a b c d e
A a B b C c D d D e
a A b B c C d D e E
a b c d e A B C D E
```

Voici un petit bout de code permettant d'afficher les caractères alphanumériques du "locale" courant dans l'ordre de ce "locale" :

```
use locale;
print +(sort grep /\w/, map { chr() } 0..255), "\n";
```

Comparez cela avec les caractères et l'ordre que vous obtenez si vous indiquez explicitement que le "locale" doit être ignoré :

```
no locale;
print +(sort grep /\w/, map { chr() } 0..255), "\n";
```

Cette ordre natif à la machine (qui est celui obtenu à moins qu'apparaisse préalablement de le bloc un `use locale`) doit être utilisé pour trier des données binaires alors que l'ordre dépendant du "locale" du premier exemple est utile pour les textes en langage naturel.

Comme indiqué dans UTILISATION DES "LOCALE", `cmp` effectue sa comparaison selon l'ordre du "locale" courant lorsque `use locale` est actif mais retombe sur une comparaison octet par octet pour les chaînes que le "locale" donne égales. Vous pouvez utiliser POSIX::strcoll() si vous ne voulez pas de cette comparaison :

```
use POSIX qw(strcoll);
$equal_in_locale =
    !strcoll("space and case ignored", "SpaceAndCaseIgnored");
```

$equal_in_locale sera vrai si l'ordre du "locale" spécifie un ordre type dictionnaire qui ignore complètement les caractères d'espacement et ne tient pas compte de la casse.

Si vous devez comparer l'égalité (au sens du "locale") d'une seule chaîne avec de nombreuses autres chaînes, vous devriez gagner un peu d'efficacité en utilisant POSIX::strxfrm() et `eq` :

```
use POSIX qw(strxfrm);
$xfrm_string = strxfrm("Mixed-case string");
print "locale collation ignores spaces\n"
    if $xfrm_string eq strxfrm("Mixed-casestring");
print "locale collation ignores hyphens\n"
    if $xfrm_string eq strxfrm("Mixedcase string");
print "locale collation ignores case\n"
    if $xfrm_string eq strxfrm("mixed-case string");
```

strxfrm() prend une chaîne et la transforme pour l'utiliser lors d'une comparaison octet par octet avec d'autres chaînes transformées. Dans les entrailles de Perl, les opérateurs Perl de comparaison dont le comportement est modifié par le "locale" appellent strxfrm() sur chacun de leurs opérandes puis effectuent une comparaison octet par octet des chaînes transformées. En appelant strxfrm() explicitement et en utilisant une comparaison ne tenant pas compte du "locale", cet exemple tente de réduire le nombre de transformation. Mais en réalité, cela ne réduit rien : la magie de Perl (voir Magic Variables in *perlguts*) crée la version transformée d'une chaîne la première fois qu'elle est nécessaire et conserve cette version au cas où elle serait à nouveau nécessaire. Un exemple réécrit tout simplement en utilisant `cmp` tournera aussi vite. Elle gère aussi correctement les caractères nuls présents dans les chaînes alors que strxfrm() considère le premier caractère nul rencontré comme étant la fin de la chaîne. N'espérez pas que les chaînes transformées soient portables d'un système à un autre – ou même d'une version à une autre d'un même système. En résumé, n'appelez pas strxfrm() directement : laissez Perl le faire pour vous.

Note : `use locale` n'est pas présent dans plusieurs exemples car il n'est pas nécessaire ; strcoll() et strxfrm() existent uniquement pour générer des résultats dépendant du "locale" par conséquent ils respectent toujours le "locale" courant `LC_COLLATE`.

5.2 Catégorie LC_CTYPE: type de caractères

Dans la portée d'un `use locale`, Perl respecte le réglage du "locale" `LC_CTYPE`. Cela contrôle la notion de caractères alphabétiques qu'utilise l'application. Cela affecte la meta-notation Perl `\w` des expressions rationnelles qui indique un caractère alphanumérique – c'est à dire un caractère alphabétique ou numérique. (Consultez *perlre* pour plus d'information sur les expressions rationnelles.) Grâce à `LC_CTYPE`, et selon les réglages de votre "locale", des caractères tels que 'æ', 'ð', 'ß' et 'ø' peuvent être considérés comme des caractères `\w`.

Le locale `LC_CTYPE` fournit aussi une table de conversion de caractères entre majuscules et minuscules. Cela affecte les fonctions de gestion de la casse – lc(), lcfirst(), uc() et ucfirst(), les interpolations dépendant de la casse `\l` `\L`, `\u` et `\U` dans les chaînes entre guillemets et les substitutions `s///` et les motifs d'expressions rationnelles indépendant de la casse grâce au modificateur `i`.

Finalement, `LC_CTYPE` affecte les fonctions de tests POSIX portant sur les classes de caractères – isalpha(), islower() et autres. Par exemple, si vous passez du "locale" "C" vers un "locale" scandinave 7-bit, vous constaterez que "|" passe de la classe ispunct() à la classe isalpha().

Note : une définition erronée ou malicieuse de `LC_CTYPE` peut amener votre application à considérer comme alphanumérique des caractères qui ne le sont pas clairement. Pour une reconnaissance stricte des lettres et des chiffres – par exemple, dans une commande – les application tenant compte des "locale" devrait utiliser `\w` à l'intérieur d'un bloc `no locale`. Voir SÉCURITÉ (§6).

5.3 Catégorie LC_NUMERIC: format numérique

Dans la portée d'un `use locale`, Perl respecte le réglage du "locale" `LC_NUMERIC` qui contrôle la manière dont une application doit présenter les nombres pour être lisible par un humain lors de leur affichage par les fonctions printf(), sprintf() et write(). La conversion d'une chaîne vers un nombre par la fonction POSIX::strtod() est aussi affectée. Dans la plupart des implémentations, le seul effet est le changement du caractère utilisé comme séparateur décimal – peut-être de '.'

en ’,’. Ces fonctions ne tiennent pas compte des raffinements tels que la séparation des milliers et autres. (Voir La fonction localeconv si vous vous intéressez à cela.)

La sortie produite par print() n’est **jamais** affecté par le "locale" courant : son comportement ne dépend pas d’un `use locale` ou d’un `no locale` et correspond à celui de printf() avec un "locale" "C". C’est la même chose pour les conversions internes de Perl entre formats de chaînes et formats numériques :

```
use POSIX qw(strtod);
use locale;

$n = 5/2;   # Affecte la valeur numérique 2.5 à $n

$a = " $n"; # conversion en chaîne indépendant du "locale"

print "half five is $n\n";       # Affichage indépendant du "locale"

printf "half five is %g\n", $n;  # Affichage dépendant du "locale"

print "DECIMAL POINT IS COMMA\n"
    if $n == (strtod("2,5"))[0]; # Conversion dépendante du "locale"
```

5.4 Catégorie LC_MONETARY: format des valeurs monétaires

Le C standard définit la catégorie `LC_MONETARY` mais aucune fonction n’est affectée par son contenu. Par conséquent, Perl n’en fera rien. Si vous voulez vraiment utiliser `LC_MONETARY`, vous pouvez demander son contenu – voir La fonction localeconv – et utiliser l’information renvoyée pour présenter vos valeurs monétaires. Par contre, vous constaterez sûrement que, aussi volumineuse et complexe que soit cette information, elle ne ne vous suffira pas : la présentation de valeurs monétaires est très difficile.

5.5 LC_TIME

La valeur produite par POSIX::strftime(), qui construit une chaîne contenant une date dans un format lisible par un être humain, est affectée par le "locale" courant `LC_TIME`. Donc, avec un "locale" français, la valeur produite par `%B` (nom complet du mois) pour le premier mois de l’année sera "janvier". Voici comment obtenir la liste des noms longs des mois du "locale" courant :

```
use POSIX qw(strftime);
for (0..11) {
    $long_month_name[$_] =
        strftime("%B", 0, 0, 0, 1, $_, 96);
}
```

Note : `use locale` n’est pas nécessaire dans cette exemple. En tant que fonction n’existant que pour générer un résultat dépendant du "locale", strftime() respecte toujours le "locale" `LC_TIME`.

5.6 Autres catégories

La dernière catégorie de "locale" `LC_MESSAGES` n’est pas actuellement utilisée par Perl – excepter qu’elle peut affecter le comportement de certaines fonctions de quelques bibliothèques appelées par des extensions ne faisant pas partie de la distribution Perl standard ou du système lui-même ou de ses utilitaires. Remarquez bien que la valeur de `$!` et les messages d’erreur produits par des utilitaires externes peuvent être modifiés par `LC_MESSAGES`. Si vous voulez des codes d’erreur portable, utilisez `%!`. Voir *Errno*.

6 SÉCURITÉ

Bien que la présentation des questions de sécurité en Perl soit faite dans *perlsec*, une présentation de la gestion par Perl des "locale" ne pourrait être complète si elle n'attirait pas votre attention sur les questions de sécurité directement liées aux "locale". Les "locale" – en particulier sur les systèmes qui autorisent les utilisateurs non privilégiés à construire leur propre "locale" – ne sont pas sûr. Un "locale" malicieux (ou tout simplement mal fait) peut amener une application liée aux "locale" à produire des résultats inattendus. Voici quelques possibilités :

- Les expressions rationnelles vérifiant la validité des noms de fichiers ou des adresses mail en utilisant `\w` peuvent être abusées par un "locale" `LC_CTYPE` qui prétend que des caractères tels que ">" et "|" sont alphanumériques.
- Les interpolations de chaînes avec des conversions majuscules/minuscules telles que `$dest = "C:\U$name.$ext"` peuvent produire de dangereux résultats si une table de conversion LC_CTYPE erronée est active.
- Certains systèmes mal conçus autorisent même les utilisateurs à modifier le "locale" "C". Si le séparateur décimal de la catégorie `LC_NUMERIC` du "locale" "C" est sournoisement changé du point vers la virgule alors `sprintf("%g", 0.123456e3)` produira le résultat "123,456". De nombreuses personnes l'interpréteront alors comme cent vingt trois mille quatre cent cinquante six.
- Un "locale" `LC_COLLATE` sournois peut amener les étudiants avec une note "D" à être classés devant ceux ayant un "A".
- Une application se reposant sur les informations de `LC_MONETARY` pourrait produire des débits comme si c'était des crédit et vice versa si son "locale" a été perverti. Ou elle pourrait faire des paiements en dollars US au lieu de dollars de Honk Kong.
- Les dates et le nom des jours dans les dates formatées par strftime() pourrait être manipulés avantageusement par un utilisateur malicieux ayant la possibilité de modifier le "locale" `LC_DATE`.

De tels dangers ne sont pas spécifiques au système de "locale" : tous les aspects de l'environnement d'une application qui peuvent être malicieusement modifiés présente les mêmes risques. De même, ils ne sont pas spécifiques à Perl : tous les langages de programmation qui vous permettent d'écrire des programmes qui tiennent compte de leur environnement sont exposés à ces problèmes.

Perl ne peut pas vous protéger de toutes les possibilités présentées dans ces exemples – il n'y a rien de mieux que votre propre vigilance – mais, lorsque `use locale` est actif, Perl utilise le mécanisme de souillure (voir *perlsec*) pour marquer les chaînes qui sont dépendantes du "locale" et qui, par conséquence, ne sont pas sûres. Voici une liste des opérateurs et fonctions dont le comportement vis-à-vis des souillures peut être affecté par le "locale" :

Opérateurs de comparaison (`lt`, `le`, `ge`, `gt` et `cmp`):

Les scalaires vrai/faux (ou plus petit/égal/plus grand) ne sont jamais souillés.

Interpolation dépendant de la casse (avec `\l`, `\L`, `\u` ou `\U`)

Les chaînes résultant d'une telle interpolation sont souillées si `use locale` est actif.

Opérateur de reconnaissance (`m//`):

Les scalaires vrai/faux ne sont jamais souillés.

Les sous-motifs délivrés soit par un résultat dans un contexte de liste soit par $1, etc. sont souillés si `use locale` est actif et si le sous-motif de l'expression rationnelle contient `\w` (pour reconnaître un caractère alphanumérique), `\W` (un caractère non alphanumériques), `\s` (un caractère blanc) ou `\S` (un caractère non blanc). Les variables $&, $‘, $’ et $+ sont aussi souillées si `use locale` est actif et si l'expression rationnelle contient `\w`, `\W`, `\s` ou `\S`.

Opérateur de substitution (`s///`):

A le même comportement que l'opérateur de reconnaissance. De plus, l'opérande de gauche d'un `=~` devient souillé lorsque `use locale` est actif et si la modification est le résultat d'une substitution basée sur une expression rationnelle impliquant `\w`, `\W`, `\s` ou `\S` ainsi que les changements de casse avec `\l`, `\L`,`\u` ou `\U`.

Fonctions de présentation (printf() et write()):

Le résultat (échec ou réussite) n'est jamais souillé.

Les fonctions de changement de casse (lc(), lcfirst(), uc(), ucfirst()):

Le résultat est souillé si `use locale` est actif.

Les fonction POSIX dépendant du "locale" (localeconv(), strcoll(), strftime(), strxfrm()):

Le résultat n'est jamais souillé.

Les tests de classes de caractères POSIX (isalnum(), isalpha(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace(), isupper(), isxdigit()):

Le résultat (vrai/faux) n'est jamais souillé.

Les trois exemples suivant illustrent les souillures dépendant du "locale".

Le premier programme, qui ignore les "locale", ne fonctionne pas : une valeur prise directement de la ligne de commande ne peut être utilisée pour nommer un fichier de sortie lorsque le mode souillé est actif.

```
#/usr/local/bin/perl -T
# mode souillé actif (-T)

# Omission de la vérification de la ligne de commande
$tainted_output_file = shift;

open(F, ">$tainted_output_file")
    or warn "Open of $untainted_output_file failed: $!\n";
```

Ce programme peut être transformé afin de "blanchir" la valeur souillée grâce à une expression rationnelle : le deuxième exemple – qui ignore encore les informations du "locale" – fonctionne et crée le fichier dont le nom est donné sur la ligne de commande si il le peut.

```
#/usr/local/bin/perl -T

$tainted_output_file = shift;
$tainted_output_file =~ m%[\w/]+%;
$untainted_output_file = $&;

open(F, ">$untainted_output_file")
    or warn "Open of $untainted_output_file failed: $!\n";
```

Comparez avec le programme suivant qui, lui, prend en compte les "locale" :

```
#/usr/local/bin/perl -T

$tainted_output_file = shift;
use locale;
$tainted_output_file =~ m%[\w/]+%;
$localized_output_file = $&;

open(F, ">$localized_output_file")
    or warn "Open of $localized_output_file failed: $!\n";
```

Ce troisième programme ne peut pas fonctionner parce que $& est souillée : c'est le résultat d'une reconnaissance impliquant `\w` alors que `use locale` est actif.

7 ENVIRONNEMENT

PERL_BADLANG

Une chaîne qui peut supprimer l'avertissement de Perl au démarrage au sujet des réglages de "locale" incorrectes. L'échec peut provenir d'un support défectueux des "locale" sur votre système – ou d'une faute de frappe dans le nom du "locale" lors de l'écriture de votre environnement. Si cette variable d'environnement est absente ou si sa valeur ne vaut pas zéro – "0" ou "" – Perl se plaindra au sujet des réglages de "locale" incorrectes.

Note : PERL_BADLANG ne vous donne qu'un moyen de cacher le message d'avertissement. Ce message vous signale un problème de votre système de "locale" et vous devriez poussez vos investigations pour savoir d'où vient ce problème.

Les variables d'environnement suivantes ne sont pas spécifiques à Perl : elles font parties de la méthode standard (ISO C, XPG4, POSIX 1.c) setlocale() permettant de contrôler la manière dont les applications gèrent les données.

LC_ALL

`LC_ALL` est la variable d'environnement "cache-tout-le-reste". Si elle existe, elle cache toutes les autres variables d'environnement liées au "locale".

LANGUAGE

NOTE : `LANGUAGE` est une extension GNU. Elle ne vous affectera que si vous utilisez la libc GNU. C'est le cas si vous utilisez Linux. Si vous utilisez un UNIX "commercial", vous n'utilisez probablement *pas* la libc GNU et vous pouvez ignorer `LANGUAGE`.

En revanche, si votre bibliothèque utilise `LANGUAGE`, cela affectera la langue des messages d'information, d'avertissement et d'erreur produits par vos commandes (autrement dit, c'est comme `LC_MESSAGES`) en ayant une priorité plus élevée que `LC_ALL`. De plus, ce n'est une simple valeur mais une suite (une liste séparée par ":") de *langues* (pas des "locale"). Voir `gettext` dans la documentation de votre bibliothèque GNU pour plus d'information.

LC_CTYPE

En l'absence de LC_ALL, LC_CTYPE détermine le type de caractères du "locale". En l'absence de LC_ALL et de LC_CTYPE, LANG est utilisé pour choisir le type de caractères.

LC_COLLATE

En l'absence de LC_ALL, LC_COLLATE détermine l'ordre (de tri) du "locale". En l'absence de LC_ALL et de LC_COLLATE, LANG est utilisé pour choir l'ordre.

LC_MONETARY

En l'absence de LC_ALL, LC_MONETARY détermine les réglages monétaires du "locale". En l'absence de LC_ALL et de LC_MONETARY, LANG est utilisé pour choir les réglages monétaires.

LC_NUMERIC

En l'absence de LC_ALL, LC_NUMERIC détermine l'affichage des nombres du "locale". En l'absence de LC_ALL et de LC_NUMERIC, LANG est utilisé pour choir l'affichage des nombres.

LC_TIME

En l'absence de LC_ALL, LC_TIME détermine l'affichage des dates et heures du "locale". En l'absence de LC_ALL et de LC_TIME, LANG est utilisé pour choir l'affichage des dates et heures.

LANG

LANG est la variable d'environnement générale des "locale". Si elle est positionnée, elle est utilisée en dernier recours après LC_ALL et LC_... spécifique à chaque catégorie.

8 NOTES

8.1 Compatibilité

Les versions de Perl antérieures à 5.004 ignorent la plupart du temps les informations de "locale" et adoptent un comportement général similaire au choix du "locale" "C" même si l'environnement du programme suggère autre chose (voir La fonction setlocale). Par défaut, Perl conserve ce comportement pour des raisons de compatibilité. Si vous voulez qu'une application Perl tienne compte aux informations de "locale", vous **devez** utiliser la directive use locale pour lui dire.

Les versions de Perl 5.002 et 5.003 utilisait l'information LC_CTYPE si elle était présente. C'est à dire que \w comprenait ce qui était une lettre au sens du "locale" indiqué par les variables d'environnement. Mais l'utilisateur n'avait aucun contrôle sur cette fonctionnalité : si la bibliothèque C supportait les "locale", Perl les utilisait.

8.2 I18N:Collate obsolète

Dans les versions de Perl antérieures à 5.004, il était possible de gérer des ordres liés aux "locale" grâce au module I18N::Collate. Ce module est maintenant devenu largement obsolète et devrait être évité dans de nouvelles applications. La fonctionnalité LC_COLLATE est maintenant complètement intégrée dans le coeur du langage Perl.

8.3 Impactes sur la vitesses des tris et sur l'utilisation de la mémoire

Comparer et trier en respectant le "locale" est normalement plus lent que le tri par défaut. Des temps deux à quatre fois plus longs ont été observés. Ce consomme aussi plus de mémoire : une fois qu'une variable scalaire de Perl à participé à une comparaison de chaîne ou à une opération de tri respectant les règles de tri du "locale", elle peut occuper de 3 à 15 fois plus de mémoire qu'avant (le facteur multiplicatif exact dépend du contenu de la chaîne, du système d'exploitation et du "locale"). Cette dégradation est plus dictée par l'implémentation des "locale" dans le système d'exploitation que par Perl.

8.4 write() et LC_NUMERIC

Les formats constituent le seul endroit où Perl utilise inconditionnellement les informations du "locale". Si l'environnement du programme spécifie un "locale" LC_NUMERIC, il est toujours utilisé pour spécifier le séparateur décimale des données sorties par format. Les sorties formatées ne peuvent pas être contrôlées par use locale car cette directive est rattachée à la structure de bloc du programme alors que, pour des raisons historiques, les formats existent en dehors de cette structure.

8.5 Définitions de "locale" disponibles librement

Vous trouverez une grosse collection de définitions de "locale" sur `ftp://dkuug.dk/i18n/WG15-collection`. Il n'y a aucune garantie de fonctionnement ni aucun support. Si votre système d'exploitation accepte l'installation de "locale" arbitraire, cela peut vous être utile soit comme "locale" directement utilisables soit comme base de développement de vos propres "locale".

8.6 I18n et l10n

"Internationalization" est souvent abrégé en **i18n** puisque la première et la dernière lettres sont séparées par 18 autres lettres. De la même manière, "localization" donne **l10n**.

8.7 Un standard imparfait

L'internationalisation, telle que définie par les standard C et POSIX, peut être considérée comme incomplète, de faible gain et d'une granularité trop grossière (les "locale" s'appliquent à l'ensemble d'un process alors qu'ils devraient pouvoir s'appliquer à un seul fil (thread), à un seul groupe de fenêtres ou autres). Elle a aussi tendance, comme dans tous les groupes de standardisation. à diviser le monde en nations alors que nous savons tous qu'il peut aussi se diviser en catégories telles les banquiers, les cyclistes, les joueurs, etc. Mais, pour l'instant, c'est le seul standard que nous ayons. Cela peut être considéré comme un bug.

9 BUGS

9.1 Les systèmes bugués

Sur certains systèmes, le support des "locale" est bugué et ne peut être corrigé ou utilisé par Perl. De telles déficiences peuvent aboutir à de mystérieux plantage de Perl lorsque `use locale` est actif. Si vous êtes confronté à un tel système, rapportez ces détails insupportables à <*perlbug@perl.com*> et plaignez-vous auprès de votre vendeur : peut-être existe-t-il des correctifs pour votre système. Parfois, ces correctifs s'appellent une mise à jour.

10 VOIR AUSSI

isalnum in *POSIX*
isalpha in *POSIX*
isdigit in *POSIX*
isgraph in *POSIX*
islower in *POSIX*
isprint in *POSIX*,
ispunct in *POSIX*
isspace in *POSIX*
isupper in *POSIX*,
isxdigit in *POSIX*
localeconv in *POSIX*
setlocale in *POSIX*,
strcoll in *POSIX*
strftime in *POSIX*
strtod in *POSIX*,
strxfrm in *POSIX*

11 HISTORIQUE

Document original *perli18n.pod* de Jarkko Hietaniemi modifié par Dominic Dunlop assisté des perl5-porters. Prose un peu améliorée par Tom Christiansen.

Dernière mise à jour : Thu Jun 11 08:44:13 MDT 1998

12 TRADUCTION

12.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.6.0. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

12.2 Traducteur

Paul Gaborit <Paul.Gaborit @ enstimac.fr>

12.3 Relecture

Personne pour l'instant.

13 À propos de ce document

Ce document est la traduction française du document orginal distribué avec perl. Vous pouvez retrouver l'ensemble de la documentation française Perl (éventuellement mise à jour) en consultant l'URL <http://perl.enstimac.fr/>.

Ce document PDF a été produit Paul Gaborit. Si vous utilisez la version PDF de cette documentation (ou une version papier issue de la version PDF) pour tout autre usage qu'un usage personnel, je vous serai reconnaissant de m'en informer par un petit message <mailto:Paul.Gaborit@enstimac.fr>.

Si vous avez des remarques concernant ce document, en premier lieu, contactez la traducteur (vous devriez trouver son adresse électronique dans la rubrique TRADUCTION) et expliquez-lui gentiment vos remarques ou critiques. Il devrait normalement vous répondre et prendre en compte votre avis. En l'absence de réponse, vous pouvez éventuellement me contacter.

Vous pouvez aussi participer à l'effort de traduction de la documentation Perl. Toutes les bonnes volontés sont les bienvenues. Vous devriez trouver tous les renseignements nécessaires en consultant l'URL ci-dessus.