RFC 8264 : PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 6 octobre 2017

Date de publication du RFC : Octobre 2017

https://www.bortzmeyer.org/8264.html

Dans la longue marche vers une plus grande **internationalisation** de l'Internet, la question des **identificateurs** (comme par exemple les noms de domaine) a toujours été délicate. Ce sont en effet à la fois des éléments techniques, traités automatiquement par les programmes, et des marqueurs d'identité, vus par les humains (par exemple sur des cartes de visite) et manipulés par eux. Plutôt que de laisser chaque protocole internationaliser ses identificateurs (plus ou moins bien), l'approche de ce RFC est unificatrice, en élaborant des règles qui peuvent servir à de larges classes d'identificateurs, pour de nombreux protocoles différents. Il remplace le premier RFC qui avait suivi cette voie, le RFC 7564 [1], désormais dépassé (mais les changements sont peu importants, c'est juste de la maintenance).

Cette opposition entre « éléments techniques » et « textes prévus pour l'utilisateur » est au cœur du RFC 2277 qui pose comme principe politique qu'on internationalise les seconds, mais pas les premiers. Cet excellent principe achoppe souvent sur la question des identificateurs, qui sont les deux à la fois. D'un côté, les programmes doivent les traiter (les identificateurs doivent donc être clairement définis, sans ambiguïté), de l'autre les humains les voient, les communiquent, les échangent (les identificateurs doivent donc permettre, sinon toutes les constructions du langage humain, en tout cas un sous-ensemble d'Unicode qui parait raisonnable aux humains ordinaires : pas question d'imposer `stephane` comme nom de "*login*" à un utilisateur nommé Stéphane, avec un accent sur le E). C'est cette double nature des identificateurs (ainsi, il est vrai, que l'énorme couche de bureaucratie qui gère les noms de domaine) qui explique la durée et la vivacité des discussions sur les IDN.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7564.txt`

Maintenant que ces IDN existent (depuis plus de quatorze ans, RFC 3490), que faire avec les autres identificateurs ? Une possibilité aurait été que chaque protocole se débrouille avec ses propres identificateurs, comme l'a fait le DNS avec les noms de domaine. Mais cela menait à une duplication du travail (et tous les auteurs de protocole ne sont pas des experts Unicode) et surtout à un risque de choix très différents : certains protocoles autoriseraient tel caractère Unicode et d'autres pas, sans que l'utilisateur ordinaire puisse comprendre clairement les raisons de cette différence. L'idée de base du groupe PRECIS <https://tools.ietf.org/wg/precis> était donc d'essayer de faire des règles qui s'appliqueraient à beaucoup de protocoles, épargnant aux concepteurs de ces protocoles de les concevoir eux-mêmes, et offrant à l'utilisateur une certaine homogénéité. Ce RFC 8264 est le cadre de définition des identificateurs internationalisés. Ce cadre permet la manipulation d'identificateurs internationalisés (par exemple leur comparaison, comme lorsqu'un utilisateur tape son nom et son mot de passe, et qu'il faut les vérifier, cf. RFC 6943.)

Certaines personnes, surtout en Anglosaxonnie, pourraient estimer que c'est bien compliqué tout cela, et qu'il vaudrait mieux limiter les identificateurs à ASCII. Certes, Unicode est compliqué, mais sa complexité ne fait que refléter celle des langues humaines (section 6.1 de notre RFC). On ne peut pas simplifier Unicode, sauf à éliminer une partie de la diversité humaine.

Le nom du groupe de travail PRECIS reflète les trois fonctions essentielles que font les programmes qui manipulent les identificateurs internationalisés :

— PR pour la **préparation** des identificateurs, les opérations préliminaires comme la conversion depuis un jeu de caractères local non-Unicode,
— E pour l'**application** ("*enforcement*") des règles, afin de s'assurer qu'une chaîne de caractères Unicode est légale ou pas pour un usage donné,
— C pour la **comparaison** entre les identificateurs, afin de déterminer leur égalité (« cette ressource n'est accessible qu'à l'utilisateur Thérèse or elle est connectée avec le nom therese »),
— Le IS final étant pour "*Internationalized Strings*".

A priori, les serveurs Internet seront responsables de l'application, les clients n'ayant à faire qu'une préparation. Ayant souvent moins de ressources, les clients pourraient en effet avoir du mal à faire certaines opérations Unicode complexes (section 3).

Les principes du travail de PRECIS sont :

— Définir un petit nombre (deux, actuellement) de classes spécifiant un jeu de caractères autorisés, classes applicables à un grand nombre d'usages.
— Définir le contenu de ces classes en fonction d'un algorithme, reposant sur les propriétés Unicode (contrairement à stringprep où le contenu des classes était énuméré dans le RFC). Ainsi, lorsqu'une nouvelle version d'Unicode sort (la version actuelle est la 10.0 <https://www.bortzmeyer.org/unicode-10-0.html>), il suffit de refaire tourner l'algorithme et on obtient le contenu à jour de la classe.
— Spécifier les classes par une inclusion : tout caractère non explicitement listé comme membre de la classe est automatiquement exclu.
— Permettre aux protocoles applicatifs de définir un profil d'une classe, à savoir une restriction de ses membres, ou bien d'autres précisions sur des sujets comme la normalisation Unicode. Il y a ainsi un profil pour les noms d'utilisateurs, un profil pour les mots de passe, etc (cf. RFC 8265). Notez qu'il est prévu que le nombre de profils reste limité, pour ne pas réintroduire l'excessive variété que voulait justement éviter PRECIS.

Si tout va bien, ces principes permettront l'indépendance vis-à-vis des versions d'Unicode (ou, plus exactement, la possibilité de passer à une nouvelle version d'Unicode sans faire une nouvelle norme incompatible), le partage des tables et du logiciel entre applications (par exemple par le biais de bibliothèques communes, utilisables si toutes les applications reposent sur PRECIS), et moins de surprises pour les utilisateurs, qui auraient été bien embêtés si chaque protocole Internet avait eu une manière complètement différente de prévoir l'internationalisation.

Bien, quelles sont donc les classes prévues par PRECIS (section 4) ? L'idée est de faire un compromis entre expressivité et sécurité. Qu'est-ce que la sécurité vient faire là dedans ? Personne ne le sait trop (le RFC utilise plusieurs fois "*safe*" et "*safety*" sans expliquer face à quels risques) mais Unicode fait souvent peur aux informaticiens anglo-saxons et il est donc courant d'estimer qu'il existe des caractères dangereux.

Il y a donc deux classes en tout et pour tout dans PRECIS : `IdentifierClass` et `FreeformClass`. La première classe servira à identifier utilisateurs, machines, pièces de discussions en messagerie instantanée, fichiers, etc, et ne permettra que les lettres, les nombres et quelques symboles (comme le ! ou le +, car ils étaient dans ASCII). C'est contraignant mais l'idée est qu'on veut des désignations simples et sans ambiguïté, pas écrire des romans. La seconde classe servira à tout le reste (mots de passe, textes affichés comme la description d'une pièce XMPP, nom complet d'une machine, etc). Par exemple, une imprimante sera `imprimante-jean-et-thérèse` pour les protocoles qui demandent un nom de la classe `IdentifierClass` et `Imprimante de Jean & Thérèse` lorsqu'on pourra utiliser un nom `FreeformClass`.

Les classes sont définies par les caractères inclus et exclus. Plus précisément, un caractère peut être, pour une classe donnée (voir aussi la section 8) :

— Valide (ou PVALID pour "*Protocol Valid*"),
— Interdit,
— Autorisé dans certains contextes seulement, c'est-à-dire que l'autorisation dépendra des caractères voisins (pas facile de trouver un exemple dans l'alphabet latin, cela sert surtout à d'autres écritures),
— Pas encore affecté dans Unicode.

La classe `IdentifierClass` se définit donc par :

— Caractères valides : les lettres et chiffres (rappelons qu'on définit des identificateurs internationalisés : on ne se limite donc pas aux lettres et chiffres latins), et les symboles traditionnels, ceux d'ASCII (comme le tiret ou le tilde).
— Caractères valides dans certains contextes : ceux de la rare catégorie `JoinControl` du RFC 5892, section 2.8.
— Caractères interdits : tout le reste (espaces, ponctuation, la plupart des symboles...).
— Les caractères non encore affectés sont également interdits.

La classe `Freeform Class` se définit, elle, par :

— Caractères valides : les lettres et chiffres (rappelons qu'on définit des identificateurs internationalisés : on ne se limite donc pas aux lettres et chiffres latins), les espaces, la ponctuation, les symboles (oui, oui, y compris les emojis comme [Caractère Unicode non montré [2]] `<https://r12a.github.io/uniview/?char=1F95E>` cf. sections 4.3.1 et 9.15).
— Caractères valides dans certains contextes : ceux de la catégorie `JoinControl` plus quelques exceptions.
— Caractères interdits : tout le reste, ce qui fait surtout les caractères de contrôle comme U+0007 (celui qui fait sonner votre ordinateur).
— Les caractères non encore affectés sont également interdits.

Ces listes de caractères autorisés ou interdits ne sont pas suffisantes. Il y a d'autres points à régler (section 5), ce qui se fait typiquement dans les profils. Ainsi, un profil doit définir :

— Normalisation Unicode à utiliser. NFC est recommandée.
— La correspondance entre les caractères et leur version large ("*width mapping*"). Est-ce que "*FULLWIDTH DIGIT ZERO*" (U+FF10) doit être considéré comme équivalent au zéro traditionnel, de largeur « normale », U+0030 ? Notez que certaines normalisations (mais qui ne sont pas celle recommandée), comme NFKC, règlent le problème. Autrement, la recommandation du RFC est « oui, il faut rendre ces caractères équivalents » car l'utilisateur serait certainement surpris que `target0` et `target[Caractère Unicode non montré ]` soient considérés différents (le fameux POLA, principe de la moindre surprise).

2. Car trop difficile à faire afficher par LaTeX

— D'autres correspondances peuvent être spécifiées par le profil (comme de transformer tous les espaces Unicode en l'espace traditionnel ASCII U+0020).
— Changement de la casse des caractères, par exemple pour tout mettre en minuscules. Si c'est décidé pour un profil, le RFC recommande que cela soit fait avec la méthode Unicode standard, `toLowerCase()` (section 3.13 de la norme Unicode). Attention, cette méthode Unicode ne gère pas des cas qui dépendent de la langue, dont le plus fameux est le i sans point du turc (U+0131 c'est-à-dire [Caractère Unicode non montré]). Le changement de casse est évidemment déconseillé pour les mots de passe (puisqu'il diminue l'entropie). Notez aussi que ce cas illustre le fait que les transformations PRECIS ne sont pas sans perte : si on met tout en minuscules, `Poussin` ne se distingue plus de `poussin`.
— Interdiction de certains mélanges de caractères de directionnalité différentes. Il y a des écritures qui vont de gauche à droite et d'autres de droite à gauche, et leur combinaison peut entrainer des surprises <`https://www.bortzmeyer.org/affichage-bidi.html`> à l'affichage. Dans certains cas, un profil peut vouloir limiter ce mélange de directionnalités.

Un profil peut également interdire certains caractères normalement autorisés (mais pas l'inverse).

Au passage, pour la comparaison, le RFC (section 7) impose un ordre à ces opérations. Par exemple, la mise en correspondance de la version large sur la version normale doit se faire avant l'éventuel changement de casse. C'est important car ces opérations ne sont pas commutatives entre elles.

Les profils sont enregistrés à l'IANA <`https://www.iana.org/assignments/precis-parameters/precis-parameters.xml#profiles`>. Le RFC met bien en garde contre leur multiplication : toute l'idée de PRECIS est d'éviter que chaque protocole ne gère l'internationalisation des identificateurs de manière différente, ce qui empêcherait la réutilisation de code, et perturberait les utilisateurs. Si on avait un cadre commun mais des dizaines de profils différents, on ne pourrait pas dire qu'on a atteint cet objectif. Par exemple, en matière d'interface utilisateur, PRECIS essaie de s'en tenir au POLA ("*Principle of Least Astonishment*") et ce principe serait certainement violé si chaque application trouvait rigolo d'interdire un caractère ou l'autre, sans raison apparente. Le RFC estime d'ailleurs (section 5.1) qu'il ne devrait y avoir idéalement que deux ou trois profils. Mais ce n'est pas possible puisque les protocoles existent déjà, avec leurs propres règles, et qu'on ne peut pas faire table rase de l'existant (tous les protocoles qui ont déjà définis des caractères interdits, comme IRC, NFS, SMTP, XMPP, iSCSI, etc).

Un petit point en passant, avant de continuer avec les applications : vous avez noté que la classe `IdentifierClass` interdit les espaces (tous les espaces Unicode, pas seulement le U+0020 d'ASCII), alors que certaines applications acceptent les espaces dans les identificateurs (par exemple, Unix les accepte sans problèmes dans les noms de fichier, Apple permet depuis longtemps de les utiliser pour nommer iTrucs et imprimantes, etc). La section 5.3 explique cette interdiction :

— Il est très difficile de distinguer tous ces espaces entre eux,
— Certains interfaces utilisateurs risquent de ne pas les afficher, menant à confondre `françoise durand` avec `françoisedurand`.

C'est embêtant (toute contrainte est embêtante) mais le compromis a semblé raisonnable au groupe PRECIS. Tant pis pour les espaces.

Passons maintenant aux questions des développeurs d'applications (section 6 du RFC). Que doivent-ils savoir pour utiliser PRECIS correctement ? Idéalement, il suffirait de lier son code aux bonnes bibliothèques bien internationalisées et tout se ferait automatiquement. En pratique, cela ne se passera pas comme ça. Sans être obligé de lire et de comprendre tout le RFC, le développeur d'applications devra quand même réflechir un peu à l'internationalisation de son programme :

— Il est très déconseillé de créer son propre profil. Non seulement c'est plus compliqué que ça n'en a l'air, mais ça risque de dérouter les utilisateurs, si votre application a des règles légèrement différentes des règles des autres applications analogues.

— Précisez bien quel partie de l'application va être responsable pour la préparation, l'application et la comparaison. Par exemple, le travail d'application sera-t-il fait par le client ou par le serveur ? Demandez-vous aussi à quel stade les applications devront avoir fait ce travail (par exemple, en cas de "*login*", avant de donner un accès).
— Définissez bien, pour chaque utilisation d'un identificateur (chaque "*slot*", dit le RFC), quel profil doit être utilisé. Par exemple, « le nom du compte doit être conforme au profil `UsernameCaseMapped` de la classe `IdentifierClass` » (cf. RFC 8265).
— Dressez la liste des caractères interdits (en plus de ceux déjà interdits par le profil) en fonction des spécificités de votre application. Par exemple, un @ est interdit dans la partie gauche d'une adresse de courrier électronique.

Sur ce dernier point, il faut noter que la frontière est mince entre « interdire plusieurs caractères normalement autorisés par le profil » et « définir un nouveau profil ». La possibilité d'interdire des caractères supplémentaires est surtout là pour s'accomoder des protocoles existants (comme dans l'exemple du courrier ci-dessus), et pour éviter d'avoir un profil par application existante.

Votre application pourra avoir besoin de constructions au-dessus des classes existantes. Par exemple, si un nom d'utilisateur, dans votre programme, peut s'écrire « Prénom Nom », il ne peut pas être une instance de la classe `IdentifierClass`, qui n'accepte pas les espaces pour la raison indiquée plus haut. Il faudra alors définir un concept « nom d'utilisateur », par exemple en le spécifiant comme composé d'une ou plusieurs instances de `IdentifierClass`, séparées par des espaces. En ABNF :

```
username   = userpart *(1*SP userpart)
userpart   = ... ; Instance d'IdentifierClass
```

La même technique peut être utilisée pour spécifier des identificateurs qui ne seraient normalement pas autorisés par `IdentifierClass` comme `stéphane@maçonnerie-générale.fr` ou `/politique/séries/Game-`

On a vu plus haut qu'un des principes de PRECIS était de définir les caractères autorisés de manière algorithmique, à partir de leur propriétés Unicode, et non pas sous la forme d'une liste figée (qu'il faudrait réviser à chaque nouvelle version d'Unicode). Les catégories de caractères utilisées par cet algorithme sont décrites en section 9. Par exemple, on y trouve :

— `LettersDigits` qui rassemble les chiffres et les lettres. (Rappelez-vous qu'on utilise Unicode : ce ne sont pas uniquement les lettres et les chiffres d'ASCII.)
— `ASCII7`, les caractères d'ASCII, à l'exception des caractères de contrôle,
— `Spaces`, tous les espaces possibles (comme le U+200A, "*HAIR SPACE*", ainsi appelé en raison de sa minceur),
— `Symbols`, les symboles, comme U+20A3 ("*FRENCH FRANC SIGN*", [Caractère Unicode non montré]) ou U+02DB ("*OGONEK*", [Caractère Unicode non montré]),
— Etc.

Plusieurs registres IANA sont nécessaires pour stocker toutes les données nécessaires à PRECIS. La section 11 les recense tous. Le plus important est le "*PRECIS Derived Property Value*", qui est recalculé à chaque version d'Unicode. Il indique pour chaque caractère s'il est autorisé ou interdit dans un identificateur PRECIS. Voici sa version pour Unicode 6.3 <`https://www.iana.org/assignments/precis-tables-6.3.0/precis-tables-6.3.0.xhtml`> (on attend avec impatience une mise à jour…).

Les deux autres registres stockent les classes <`https://www.iana.org/assignments/precis-parameters/precis-parameters.xml#base-classes`> et les profils <`https://www.iana.org/assignments/precis-parameters/precis-parameters.xml#profiles`> (pour l'instant, ils sont quatre). Les règles d'enregistrement (section 11) dans le premier sont strictes (un RFC est nécessaire) et celles dans le second plus ouvertes (un examen par un expert est nécessaire). La section 10 explique aux experts en question ce qu'ils devront bien regarder. Elle note que l'informaticien ordinaire est en général très

ignorant des subtilités d'Unicode et des exigences de l'internationalisation, et que l'expert doit donc se montrer plus intrusif que d'habitude, en n'hésitant pas à mettre en cause les propositions qu'il reçoit. Dans beaucoup de RFC, les directives aux experts sont d'accepter, par défaut, les propositions, sauf s'il existe une bonne raison de les rejeter. Ici, c'est le contraire : le RFC recommande notamment de rejeter les profils proposés, sauf s'il y a une bonne raison de les accepter.

La section 12 est consacrée aux problèmes de sécurité qui, comme toujours lorsqu'il s'agit d'Unicode <https://www.bortzmeyer.org/idn-et-phishing.html>, sont plus imaginaires que réels. Un des problèmes envisagés est celui du risque de confusion entre deux caractères qui sont visuellement proches. Le problème existe déjà avec le seul alphabet latin (vous voyez du premier coup la différence entre google.com et goog1e.com?) mais est souvent utilisé comme prétexte pour retarder le déploiement d'Unicode. PRECIS se voulant fidèle au principe POLA, le risque de confusion est considéré comme important. Notez que le risque réel dépend de la police utilisée. Unicode normalisant des caractères et non pas des glyphes, il n'y a pas de solution générale à ce problème dans Unicode (les écritures humaines sont compliquées : c'est ainsi). Si le texte [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré] ressemble à STPETER, c'est que vous utilisez une police qui ne distingue pas tellement l'alphabet cherokee de l'alphabet latin. Est-ce que ça a des conséquences pratiques ? Le RFC cite le risque accru de hameçonnage, sans citer les nombreuses études qui montrent le contraire (cf. le "*Unicode Technical Report 36*" <http://unicode.org/reports/tr36/>, section 2, « "*the use of visually confusable characters in spoofing is often overstated*" », et la FAQ de sécurité d'Unicode <http://www.unicode.org/faq/security.html>).

Quelques conseils aux développeurs concluent cette partie : limiter le nombre de caractères ou d'écritures qu'on accepte, interdire le mélange des écritures (conseil inapplicable : dans la plupart des alphabets non-latins, on utilise des mots entiers en alphabet latin)... Le RFC conseille aussi de marquer visuellement les identificateurs qui utilisent plusieurs écritures (par exemple en utilisant des couleurs différentes), pour avertir l'utilisateur.

C'est au nom de ce principe POLA que la classe IdentifierClass est restreinte à un ensemble « sûr » de caractères (notez que ce terme « sûr » n'est jamais expliqué ou défini dans ce RFC). Comme son nom l'indique, FreeformClass est bien plus large et peut donc susciter davantage de surprises.

PRECIS gère aussi le cas des mots de passe en Unicode. Un bon mot de passe doit être difficile à deviner ou à trouver par force brute (il doit avoir beaucoup d'**entropie**). Et il faut minimiser les risques de faux positifs (un mot de passe accepté alors qu'il ne devrait pas : par exemple, des mots de passe insensibles à la casse seraient agréables pour les utilisateurs mais augmenteraient le risque de faux positifs). L'argument de l'entropie fait que le RFC déconseille de créer des profils restreints de FreeformClass, par exemple en excluant des catégories comme la ponctuation. Unicode permet des mots de passe vraiment résistants à la force brute, comme « [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré] »... D'un autre côté, comme le montre l'exemple hypothétique de mots de passe insensibles à la casse, il y a toujours une tension entre la sécurité et l'utilisabilité. Laisser les utilisateurs choisir des mots de passe comportant des caractères « exotiques » peut poser bien des problèmes par la suite lorsqu'utilisateur tentera de les taper sur un clavier peu familier. Il faut noter aussi que les mots de passe passent parfois par des protocoles intermédiaires (comme SASL, RFC 4422, ou comme RADIUS, RFC 2865) et qu'il vaut donc mieux que tout le monde utilise les mêmes règles pour éviter des surprises (comme un mot de passe refusé par un protocole intermédiaire).

Enfin, la section 13 de notre RFC se penche sur l'interopérabilité. Elle rappelle qu'UTF-8 est l'encodage recommandé (mais PRECIS est un cadre, pas un protocole complet, et un protocole conforme à PRECIS