Groupe de travail Réseau
Request for Comments : 2640
RFC mise à jour : 959
Catégorie : Proposition de norme

B. Curtin
Defense Information Systems Agency
juillet 1999
Traduction Claude Brière de L'Isle

Internationalisation du protocole de transfert de fichier

Statut du présent mémoire
Le présent document spécifie un protocole en cours de normalisation de l'Internet pour la communauté de l'Internet, et appelle à des discussions et suggestions pour son amélioration. Prière de se référer à l'édition en cours des "Normes officielles des protocoles de l'Internet" (STD 1) pour connaître l'état de la normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.

Résumé
Le protocole de transfert de fichier, tel que défini dans la [RFC959] et à la Section 4 de la [RFC1123], est un des protocoles les plus anciens et les plus largement utilisés de l'Internet. Le jeu de caractères principal du protocole, l'ASCII à 7 bits, a bien servi le protocole à travers les premières années de croissance de l'Internet. Cependant, comme l'Internet devient plus mondial, le besoin se fait sentir d'une prise en charge d'autres jeux de caractères que l'ASCII à 7 bits.

Le présent document traite de l'internationalisation (I18n) de FTP, qui inclut la prise en charge de plusieurs jeux de caractères et langages qui ont cours à travers la communauté de l'Internet. Cela est fait en étendant la spécification FTP et en donnant des recommandations pour une prise en charge appropriée de l'internationalisation.

Table des Matières

1. Introduction

Avec la croissance de l'Internet à travers le monde, l'exigence de prise en charge de jeux de caractères autres que les jeux de caractères ASCII [ASCII] / Latin-1 [ISO-8859] devient de plus en plus urgente. Pour FTP, à cause de la large base

installée, il est de la plus haute importance que ceci soit fait sans dommages pour les clients et serveurs existants. Le présent document traite de ce besoin. En le faisant, il définit une solution qui permettra quand même à la base installée d'interopérer avec les nouveaux clients et serveurs.

Le présent document améliore les capacités du protocole de transfert de fichier (FTP, *File Transfer Protocol*) en supprimant les restrictions à 7 bits sur les noms de chemins utilisés dans les commandes de client et les réponses de serveur, il RECOMMANDE l'utilisation d'un jeu de caractères universel (UCS, *Universal Character Set*) [ISO-10646], il RECOMMANDE un format de transformation UCS (UTF, *UCS Transformation Format*) [UTF-8], et définit une nouvelle commande pour la négociation de langage.

Les recommandations faites dans le présent document sont cohérentes avec celles exprimées par la politique de l'IETF au sujet des jeux de caractères et des langages telle que définie dans la [RFC2277].

1.1 Spécification des exigences

Les mots clé "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" dans ce document sont à interpréter comme décrit dans la [RFC2119].

2. Internationalisation

Le protocole de transfert de fichiers a été développé alors que les jeux de caractères prédominants étaient l'ASCII à 7 bits et l'EBCDIC à 8 bits. Aujourd'hui, ces jeux de caractères ne peuvent pas prendre en charge la large gamme de caractères nécessaires pour les systèmes multinationaux. Étant donné le nombre de jeux de caractères d'usage courant qui fournissent plus de caractères que l'ASCII à 7 bits, il est raisonnable de décider d'une façon convenable pour représenter l'union de ces possibilités. Travailler au niveau mondial exige soit la prise en charge d'un certain nombre de jeux de caractères et d'être capable de les convertir entre eux, soit d'utiliser un seul jeu de caractères préféré. Pour assurer l'interopérabilité mondiale, le présent document RECOMMANDE cette dernière approche et définit un seul jeu de caractères, en plus du NVT ASCII et de l'EBCDIC, qui sont compréhensibles par tous les systèmes. Pour FTP, ce jeu de caractères DEVRA être l'ISO/CEI 10646:1993. Pour la prise en charge de la compatibilité mondiale, il est FORTEMENT RECOMMANDÉ que les clients et serveurs utilisent le codage UTF-8 lors de l'échange de noms de chemins. Les clients et les serveurs ne sont cependant nullement obligés d'effectuer une conversion sur le contenu d'un fichier pour des opérations telles que STOR ou RETR.

Le jeu de caractères utilisé pour mémoriser des fichiers DEVRA rester une décision locale et PEUT dépendre de la capacité des systèmes d'exploitation locaux. Avant l'échange des noms de chemins, ils DEVRAIENT être convertis en format ISO/CEI 10646 et codés en UTF-8. Cette approche, tout en permettant des échanges internationaux de noms de chemins, va quand même permettre la rétro compatibilité avec les systèmes plus anciens parce que les positions des codets pour les caractères ASCII sont identiques à la séquence d'un octet en UTF-8.

Les paragraphes 2.1 et 2.2 donnent une brève description du jeu de caractères international et du codage de transfert RECOMMANDÉ dans ce document. On trouvera une description plus détaillée de l'UTF-8, de ISO/CEI 10646, et de UNICODE [UNICODE] que celle donnée dans le présent document, dans la [RFC2279].

2.1 Jeu de caractères international

Le jeu de caractères défini pour la prise en charge internationale de FTP DEVRA être le jeu de caractères universel tel que défini et amendé par la norme internationale ISO 10646:1993. Cette norme incorpore les jeux de caractères de nombreuses normes internationales, nationales, et d'entreprises existantes. La norme ISO/CEI 10646 définit deux formes de remplacement pour le codage, UCS-4 et UCS-2. UCS-4 est un codage sur quatre octets (31 bits) contenant 2**31 positions de code divisées en 128 groupes de 256 plans. Chaque plan consiste en 256 rangées de 256 cellules. UCS-2 est un jeu de caractères de 2 octets (16 bits) consistant en un plan zéro ou tableau multilingue de base (BMP, *Basic Multilingual Plane*). Actuellement, aucun jeu de caractères n'a été défini en dehors du BMP sur 2 octets.

La norme Unicode version 2.0 [UNICODE] est cohérente avec le sous-ensemble UCS-2 de ISO/CEI 10646. La norme Unicode version 2.0 comporte le répertoire des caractères IS 10646, les amendements 1 à 7 de IS 10646, et des corrections rédactionnelles et techniques.

2.2 Codage de transfert

Le format de transformation UCS 8 (UTF-8), appelé autrefois UTF-2 ou UTF-FSS, DEVRA être utilisé comme codage de

transfert pour transmettre le jeu de caractères international. UTF-8 est un codage sûr de fichier qui évite l'utilisation des valeurs d'octet qui ont une signification particulière durant l'analyse des chaînes de caractères des noms de chemins. UTF-8 est un codage sur 8 bits des caractères dans l'UCS. Un des avantages de UTF-8 est qu'il est compatible avec l'ASCII à 7 bits, de sorte qu'il n'affecte pas les programmes qui donnent une significations particulière à divers caractères ASCII ; il est prémuni contre les erreurs de synchronisation ; ses règles de codage permettent une identification facile, et il possède un espace suffisant pour prendre en charge un grand nombre de jeux de caractères.

Le codage UTF-8 représente chaque caractère UCS comme une séquence de 1 à 6 octets . Pour toutes les séquences d'un octet, le bit de poids fort est ZÉRO. Pour toute séquence de plus d'un octet, le nombre de bits UN dans le premier octet, commençant à la position du bit de poids fort, indique le nombre d'octets dans la séquence UTF-8, suivi par un bit ZÉRO. Par exemple, le premier octet d'une séquence UTF-8 de trois octets aurait 1110 comme bits de poids fort. Chaque octet supplémentaire (les octets de continuation) dans la séquence UTF-8, contient un bit UN suivi par un bit ZÉRO comme bits de poids fort. Les positions binaires restantes dans les octets de continuation sont utilisées pour identifier les caractères dans UCS. Les relations entre UCS et UTF-8 sont montrées dans le tableau suivant :

gamme UCS-4 (hex)	**séquence d'octets UTF-8 (binaire)**
00000000 - 0000007F	0xxxxxxx
00000080 - 000007FF	110xxxxx 10xxxxxx
00000800 - 0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
00010000 - 001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
00200000 - 03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
04000000 - 7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Une propriété avantageuse de l'UTF-8 est que sa séquence d'un seul octet est cohérente avec le jeu de caractères ASCII. Cette caractéristique va permettre la transition lorsque des clients qui ne prennent en charge que le vieil ASCII pourront encore interopérer avec les nouveaux serveurs qui prennent en charge le codage UTF-8.

Une autre caractéristique est que les règles de codage rendent très improbable qu'une séquence de caractères provenant d'un jeu de caractères différent puisse être confondue avec une séquence de caractères codée en UTF-8. Les clients et serveurs peuvent utiliser un sous-programme simple pour déterminer si le jeu de caractères échangé est de l'UTF-8 valide. Le paragraphe B.1 donne un exemple de code de cette vérification.

3. Noms de chemin

3.1 Conformité générale

- La restriction à 7 bits pour l'échange de noms de chemins est abandonnée.

- De nombreux systèmes d'exploitation permettent d'utiliser les caractères espace <SP>, retour chariot <CR>, et saut à la ligne <LF> au titre du nom de chemin. L'échange des noms de chemin avec ces caractères de commande particuliers va causer une mauvaise analyse des noms de chemin. C'est pourquoi les commandes ftp associées aux noms de chemins sont de la forme :

  COMMANDE <SP> <nom_de_chemin> <CRLF>.

Pour permettre l'échange de noms de chemins qui contiennent ces caractères, la définition du nom de chemin est changée de : <nom_de_chemin> ::= <chaîne> ; en format BNF
en
nom_de_chemin = 1*(%x01..%xFF) ; en format ABNF [RFC2234].

Pour éviter de confondre ces caractères au sein des noms de chemins avec des caractères de commandes spéciaux, les règles suivantes s'appliqueront :

Il DOIT y avoir un seul <SP> entre une commande ftp et le nom de chemin. Les mises en œuvre DOIVENT supposer que les caractères <SP> qui suivent le <SP> initial font partie du nom de chemin. Par exemple, le nom de chemin dans STOR <SP><SP><SP>foo.bar<CRLF> est <SP><SP>foo.bar.

Les mises en œuvre actuelles, qui peuvent permettre plusieurs caractères <SP> comme séparateurs entre la commande et le nom de chemin, DOIVENT s'assurer qu'elles se conforment à la convention du <SP> unique. Noter que les mises en œuvre qui traitent des commandes de 3 caractères (par exemple, CWD, MKD, etc.) comme une commande fixe de 4 caractères en bourrant la commande avec un <SP> en queue ne sont pas conformes à la présente spécification.

Lorsque un caractère <CR> se rencontre au titre d'un nom de chemin, il DOIT être bourré avec un caractère <NUL> avant d'envoyer la commande. À réception d'un nom de chemin qui contient une séquence <CR><NUL>, le caractère <NUL> DOIT être supprimé. Cette approche est décrite dans le protocole Telnet [RFC854] aux pages 11 et 12. Par exemple, pour mémoriser un nom de chemin foo<CR><LF>boo.bar, le nom de chemin va devenir foo<CR><NUL><LF>boo.bar avant d'envoyer la commande STOR <SP>foo<CR><NUL><LF>boo.bar<CRLF>. À réception du nom de chemin altéré, le caractère <NUL> suivant le <CR> serait supprimé pour former le nom de chemin original.

- Les clients et serveurs conformes DOIVENT prendre en charge UTF-8 pour le transfert et la réception des noms de chemin. Les clients et serveurs PEUVENT de plus donner aux utilisateurs le choix de spécifier l'interprétation des noms de chemins dans un autre codage. Noter que configurer clients et serveurs à utiliser des jeux de caractères/codages autres que UTF-8 sort du domaine d'application du présent document. Bien qu'il soit reconnu que dans certains scénarios de fonctionnement cela puisse être désirable, cela reste une question de qualité de mise en œuvre et de fonctionnement.

- Les noms de chemin sont des séquences d'octets. Le codage des noms qui sont des séquences UTF-8 valides est supposé être UTF-8. Le jeu de caractères des autres noms est indéfini. Les clients et serveurs, sauf configurés autrement pour prendre en charge un jeu de caractère natif spécifique, DOIVENT chercher une séquence d'octets UTF-8 valide pour déterminer si le nom de chemin présenté est en UTF-8.

- Pour éviter des pertes de données, les clients et serveurs DEVRAIENT utiliser des noms de chemin codés en UTF-8 lorsque ils ne sont pas capables de les convertir en un jeu de code utilisable.

- Il peut y avoir des cas où le jeu de code/codage présenté au serveur ou client ne peut pas être déterminé. Dans de tels cas, les octets bruts DEVRAIENT être utilisés.

3.2 Conformité des serveurs

- Les serveurs DOIVENT prendre en charge le dispositif UTF-8 en réponse à la commande FEAT [RFC2389]. Le dispositif UTF-8 est une ligne contenant la chaîne "UTF8" exacte. Cette chaîne n'est pas sensible à la casse, mais DEVRAIT être transmise en majuscules. La réponse à une commande FEAT DEVRAIT être :

```
C> feat
S> 211- <tout texte descriptif>
S>  ...
S>  UTF8
S>  ...
S> 211 end
```

Les points de suspension indiquent des fourre-tout où d'autres caractéristiques peuvent être incluses, mais NE SONT PAS EXIGÉES. Le retrait d'espace des lignes de caractéristiques est obligatoire [RFC2389].

- Les serveurs miroir peuvent vouloir refléter exactement le site dont ils sont le miroir. Dans de tels cas, les serveurs PEUVENT mémoriser et présenter les octets exacts du nom de chemin qu'ils ont reçu du serveur principal.

3.3 Conformité des clients

- Les clients qui n'exigent pas l'affichage du nom de chemin n'ont aucune obligation de le faire. Les clients qui n'affichent pas n'ont pas besoin de se conformer aux exigences associées à l'affichage.

- Les clients à qui le serveur présente des nom de chemin UTF-8 DEVRAIENT analyser correctement l'UTF-8 et essayer d'afficher le nom de chemin dans les limites des ressources disponibles.

- les clients DOIVENT prendre en charge la commande FEAT et reconnaître le dispositif "UTF8" (défini en 3.2 ci-dessus) pour déterminer si un serveur prend en charge le codage UTF-8.

- La sémantique des caractères des autres noms devra rester indéfinie. Si un client détecte qu'un serveur est non UTF-8, il DEVRAIT changer son affichage de façon appropriée. Comment une mise en œuvre de client traite le non UTF-8 est une question qui relève de la mise en œuvre. Elle PEUT essayer de supposer d'autres codages, donner à l'utilisateur une chance d'essayer de supposer quelque chose, ou sauvegarder les hypothèses de codage pour un serveur d'une session FTP à l'autre.

- Le rendu des glyphes sort du domaine d'application du présent document. Comment un client présente les caractères qu'il ne peut pas afficher est une question qui relève de la mise en œuvre. Le présent document RECOMMANDE que

les octets qui correspondent à des caractères non affichables soient présentés en format URL %HH défini dans la [RFC1738]. Il PEUT cependant les afficher comme points d'interrogation, avec leur valeur hexadécimale UCS, ou de toute autre façon convenable.

- De nombreux clients existants interprètent les noms de chemin de 8 bits comme étant dans le jeu de caractères local. Ils PEUVENT continuer à le faire pour les noms de chemin qui ne sont pas de l'UTF-8 valide.

4. Prise en charge du langage

Le rapport de l'atelier sur les jeux de caractères [RFC2130] suggère que les clients et les serveurs DEVRAIENT négocier un langage pour les "accueils" et les "messages d'erreur". La présente spécification interprète l'utilisation du terme "message d'erreur" de la RFC2130, comme signifiant toute chaîne de texte explicatif retournée par le serveur-PI en réponse à une commande d'utilisateur-PI.

Les mises en œuvre DEVRAIENT noter que les commandes et les réponses numériques FTP sont des éléments du protocole. À ce titre, leur utilisation n'est pas affectée par les indications exprimées par la présente spécification.

La prise en charge par le langage de l'accueil et des réponses aux commandes devra être le langage par défaut pris en charge par le serveur ou le langage accepté par le serveur et choisi par le client.

Il se peut que la réalisation de la prise en charge du langage se fasse à travers un hôte virtuel comme décrit dans [MLST]. Cependant, un serveur FTP pourrait ne pas prendre en charge les serveurs virtuels, ou les serveurs virtuels pourraient être configurés pour prendre en charge un environnement sans égard pour le langage. Pour permettre la négociation du langage, la présente spécification définit une nouvelle commande LANG. Les clients et serveurs qui se conforment à la présente spécification DOIVENT prendre en charge la commande LANG.

4.1 Commande LANG

Une nouvelle commande "LANG" est ajoutée à l'ensemble de commandes FTP pour permettre au processus serveur FTP de déterminer dans quel langage présenter l'accueil du serveur et la partie textuelle des réponses aux commandes. Le paramètre associé à la commande LANG DEVRA être une des étiquettes de langue définies dans la [RFC1766]. Si une commande LANG sans paramètre est produite, on utilisera le langage par défaut du serveur.

Les accueils et réponses produites avant la négociation de langage DEVRONT être dans le langage par défaut du serveur. Le paragraphe 4.5 de la [RFC2277] déclare que ce "langage par défaut DOIT être compréhensible par une personne de langue anglaise". Cette spécification RECOMMANDE que le langage par défaut du serveur soit l'anglais codé en utilisant l'ASCII. Ce texte peut être augmenté par du texte d'autres langages. Une fois négocié, le serveur-PI DOIT retourner les messages de serveur et la partie textuelle des réponses de commandes dans le langage négocié et codé en UTF-8. Le serveur-PI PEUT souhaiter envoyer à nouveau les messages de serveur précédemment produits dans le langage nouvellement négocié.

La commande LANG n'affecte que la présentation des messages d'accueil et du texte explicatif associé aux réponses de commandes. Aucune tentative ne devrait être faite par le serveur pour traduire les éléments de protocole (commandes FTP et réponses numériques) ou les données transmises sur la connexion de données.

L'usager-PI PEUT produire la commande LANG à tout moment durant une session FTP. Afin de tirer le meilleur parti de cette commande, elle DEVRAIT être présentée avant l'authentification. En général, elle sera produite après la commande HOST [MLST]. Noter que la production d'une commande HOST ou REIN [RFC959] va nier l'effet de la commande LANG. L'usager-PI DEVRAIT être capable de prendre en charge le codage UTF-8 pour le langage négocié. Les lignes directrices sur l'interprétation et le rendu de l'UTF-8, définies à la section 3, DEVRONT s'appliquer.

Bien que NON EXIGÉ par la présente spécification, un usager-PI DEVRAIT produire une commande FEAT [RFC2389] avant une commande LANG. Cela va permettre à l'usager-PI de déterminer si le serveur prend en charge la commande LANG et quelles options de langage.

Afin d'aider le serveur à identifier si une connexion a été établie avec un client qui se conforme à la présente spécification ou un client plus ancien, l'usager-PI DOIT envoyer une commande HOST [MLST] et/ou LANG avant de produire toute autre commande (autre que FEAT [RFC2389]). Si l'usager-PI produit une commande HOST, et si le langage par défaut du serveur est acceptable, il n'a pas besoin de produire une commande LANG. Cependant, si la mise en œuvre ne prend pas en charge la commande HOST, une commande LANG DOIT être produite. Jusqu'à ce que le serveur-PI se présente avec une commande HOST ou LANG, il DEVRAIT supposer que l'usager-PI ne se conforme pas à la présente spécification.

4.2 Syntaxe de la commande LANG

La commande LANG est définie comme suit :

```
lang-command            = "Lang" [(SP lang-tag)] CRLF
lang-tag                = Primary-tag *( "-" Sub-tag)
Primary-tag             = 1*8ALPHA
Sub-tag                 = 1*8ALPHA

lang-response           = lang-ok / error-response
lang-ok                 = "200" [SP *(%x00..%xFF) ] CRLF
error-response          = command-unrecognized / bad-argument / not-implemented / unsupported-parameter
command-unrecognized    = "500" [SP *(%x01..%xFF) ] CRLF
bad-argument            = "501" [SP *(%x01..%xFF) ] CRLF
not-implemented         = "502" [SP *(%x01..%xFF) ] CRLF
unsupported-parameter   = "504" [SP *(%x01..%xFF) ] CRLF
```

Le mot de commande "lang" est indépendant de la casse et peut être spécifié dans toute casse de caractères désirée. Donc, "LANG", "lang", "Lang", et "lAnG" sont des commandes équivalentes.

Le "Lang-tag" FACULTATIF donné comme paramètre spécifie les principales étiquettes de langage et zéro, une ou plusieurs sous étiquettes, comme défini dans la [RFC1766]. Comme décrit dans la [RFC1766] les étiquettes de langue sont traitées comme insensibles à la casse. Si elles sont omises, le serveur-PI DOIT utiliser le langage par défaut du serveur. Le serveur FTP répond à la commande "Lang" soit par "lang-ok", soit par "error-response". "lang-ok" DOIT être envoyé si le serveur FTP prend en charge la commande "Lang" et peut prendre en charge une forme de la "lang-tag". La prise en charge DEVRAIT être comme suit :

- Si le serveur-FTP reçoit "Lang" sans paramètre, il DEVRAIT retourner les messages et les réponses de commande dans le langage par défaut du serveur.

- Si le serveur-FTP reçoit "Lang" avec seulement un argument d'étiquette primaire (par exemple, en, fr, de, ja, zh, etc.), qu'il peut prendre en charge, il DEVRAIT retourner des messages et des réponses de commande dans le langage associé à cette étiquette primaire. Il est possible que le serveur-FTP prenne seulement en charge l'étiquette primaire lorsque elle est combinée avec une sous-étiquette (par exemple, en-US, en-UK, etc.). Dans un tel cas, le serveur-FTP PEUT déterminer la variante appropriée à utiliser durant la session. Comment le serveur-FTP fait cette détermination est en dehors du domaine d'application de la présente spécification. Si le serveur-FTP ne peut pas déterminer si une variante de sous-étiquette est appropriée, il DEVRAIT retourner une réponse "paramètre non pris en charge " (504).

- Si le serveur-FTP reçoit "Lang" avec une étiquette primaire et un ou des arguments de sous-étiquettes, qu'il met en œuvre, il DEVRAIT retourner des messages et des réponses de commandes à l'appui de l'argument de langage. Il est possible que le serveur-FTP puisse prendre en charge l'étiquette primaire de l'argument "Lang" mais pas la ou les sous-étiquettes. Dans ce cas, le serveur-FTP PEUT retourner des messages et réponses de commande dans la variante la plus appropriée de l'étiquette primaire qui a été mise en œuvre. Comment le serveur-FTP fait cette détermination sort du domaine d'application de la présente spécification. Si le serveur-FTP ne peut pas déterminer si une variante de sous-étiquette est appropriée, il DEVRAIT retourner une réponse "paramètre non pris en charge" (504).

Par exemple, si le client-FTP envoie une commande "LANG en-AU" et si le serveur-FTP a mis en œuvre les étiquettes de langue en-US et en-UK, il peut décider que l'étiquette de langue la plus appropriée est en-UK et retourner "200 en-AU non pris en charge. Langage réglé à en-UK". La réponse numérique est un élément de protocole et ne peut pas être changée. La chaîne associée est seulement à des fins d'illustration.

Les clients et serveurs qui se conforment à la présente spécification DOIVENT prendre en charge la commande LANG. Les clients DEVRAIENT, cependant, anticiper la réception d'une réponse de commande 500 ou 502, dans les cas où des serveurs anciens ou non conformes ne reconnaissent pas ou n'ont pas mis en œuvre "Lang". Une réponse 501 DEVRAIT être envoyée si l'argument à la commande "Lang" n'est pas syntaxiquement correct. Une réponse 504 DEVRAIT être envoyée si l'argument "Lang", tout en étant syntaxiquement correct, n'est pas mis en œuvre. Comme on l'a noté plus haut, un argument peut être considéré comme correspondant lexigraphiquement même si il n'a pas une correspondance syntaxique exacte.

4.3 Réponse Feat à la commande LANG

Un processus de serveur-FTP qui prend en charge la commande LANG, et prend en charge un langage pour les messages et réponses de commandes, DOIT inclure dans la réponse à la commande FEAT de la [RFC2389] une ligne de caractérisation qui indique que la commande LANG est prise en charge et une liste des étiquettes des langues prises en charge. Une réponse à une commande FEAT DEVRA être dans le format suivant :

```
Lang-feat     = SP "LANG" SP lang-fact CRLF
lang-fact     = lang-tag ["*"] *(";" lang-tag ["*"])

lang-tag      = Primary-tag *( "-" Sub-tag)
Primary-tag   = 1*8ALPHA
Sub-tag       = 1*8ALPHA
```

La réponse lang-feat contient la chaîne "LANG" suivie par un lang-fact. Cette chaîne n'est pas sensible à la casse, mais DEVRAIT être transmise en majuscules, comme recommandé dans la [RFC2389]. L'espace initiale indiquée dans la réponse Lang-feat est EXIGÉE par la commande FEAT. Ce DOIT être un seul caractère espace. Plus ou moins de caractères espace ne sont pas permis. Le lang-fact DEVRA comporter les lang-tags que le serveur-FTP peut prendre en charge. Au moins une lang-tag DOIT être incluse dans la réponse FEAT. La lang-tag DEVRA être sous la forme décrite plus haut dans ce document. L'astérisque FACULTATIVE, lorsque elle est présente, DEVRA indiquer la lang-tag actuelle utilisée par le serveur-FTP pour les messages et les réponses.

4.3.1 Exemples de feat

```
C> feat
S> 211- <tout texte descriptif>
S>  ...
S>  LANG EN*
S>  ...
S> 211 fin
```

Dans cet exemple, le serveur-FTP peut seulement prendre en charge l'anglais, qui est la langue actuelle (comme le montre l'astérisque) utilisée par le serveur pour les messages et les réponses de commandes.

```
C> feat
S> 211- <tout texte descriptif>
S>  ...
S>  LANG EN*;FR
S>  ...
S> 211 fin
C> LANG fr
S> 200 La réponse sera changée en français

C> feat
S> 211- <tout texte descriptif>
S>  ...
S>  LANG EN;FR*
S>  ...
S> 211 fin
```

Dans cet exemple, le serveur-FTP prend en charge le français et l'anglais, comme le montre la réponse initiale à la commande FEAT. L'astérisque indique que l'anglais est la langue actuelle utilisée par le serveur-FTP. Après la production d'une commande LANG pour changer le langage en français, la réponse FEAT montre que le français est le langage courant utilisé.

Dans les exemples ci-dessus les points de suspension indiquent des fourre-tout où d'autres caractéristiques peuvent être incluses, mais NE SONT PAS EXIGÉES.

5. Considérations pour la sécurité

Le présent document traite de la prise en charge des jeux de caractères au delà d'un octet et d'une nouvelle commande de négociation de langage. La conformité au présent document ne devrait pas induire de risque pour la sécurité.

6. Remerciements

Les personnes suivantes ont contribué au présent document : D. J. Bernstein, Martin J. Duerst, Mark Harris, Paul Hethmon, Alun Jones, Gregory Lundberg, James Matthews, Keith Moore, Sandra O'Donnell, Benjamin Riefenstahl, Stephen Tihor (et d'autres du groupe de travail FTPEXT).

7. Glossaire

BIDI – abréviation de bidirectionnel, référence à du texte mêlé de droite à gauche et de gauche à droite.

Jeu de caractères – collection de caractères utilisés pour représenter les informations textuelles dans lesquelles chaque caractère a une valeur numérique.

Jeu de code – (voir jeu de caractères).

Glyphe – image de caractère représentée sur un appareil d'affichage.

I18N – "I dix huit N", la première et la dernière lettre du mot "internationalisation" et les dix huit lettres qui sont entre.

UCS-2 – forme du jeu de caractères à deux octets de la norme ISO/CEI 10646.

UCS-4 – forme du jeu de caractères à quatre octets de la norme ISO/CEI 10646.

UTF-8 – Format de transformation UCS représenté en 8 bits.

TF-16 – Format de 16 bits qui inclut le BMP (codé directement) et les paires suppléantes pour représenter les caractères dans les plans 01 à 16; équivalents à Unicode.

8. Bibliographie

[ASCII] ANSI X3.4:1986 Coded Character Sets - 7 Bit American National Standard Code for Information Interchange (7-bit ASCII)

[ISO-8859] ISO 8859. International standard -- Information processing -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1 (1987) -- Part 2: Latin alphabet No. 2 (1987) -- Part 3: Latin alphabet No. 3 (1988) -- Part 4: Latin alphabet No. 4 (1988) -- Part 5: Latin/Cyrillic alphabet (1988) -- Part 6: Latin/Arabic alphabet (1987) -- Part : Latin/Greek alphabet (1987) -- Part 8: Latin/Hebrew alphabet (1988) -- Part 9: Latin alphabet No. 5 (1989) -- Part10: Latin alphabet No. 6 (1992)

[ISO-10646] ISO/CEI 10646-1:1993. International standard -- Information technology -- Universal multiple-octet coded character set (UCS) -- Part 1: Architecture and basic multilingual plane.

[RFC0854] J. Postel et J. Reynolds,"Spécification du protocole TELNET", STD 8, mai 1983.

[RFC0959] J. Postel et J. Reynolds, "Protocole de transfert de fichiers (FTP)", STD 9, octobre 1985.

[RFC1123] R. Braden, éditeur, "Exigences pour les hôtes Internet – Application et prise en charge", STD 3, octobre 1989.

[RFC1738] T. Berners-Lee et autres, "Localisateurs uniformes de ressource (URL)", décembre 1994. *(P.S., Obsolète, voir les* RFC4248 *et* 4266*)*

[RFC1766] H. Alvestrand, "Étiquettes pour l'identification des langues", mars 1995. *(Obs., voir* RFC3066, RFC3282)

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997.

[RFC2130] C. Weider, C. Preston, K. Simonsen, H. Alvestrand, R. Atkinson, M. Crispin, P. Svanberg, "Rapport de l'atelier Jeux de caractères de l'IAB tenu du 29 février au 1er mars 1996", avril 1997. *(Information)*

[RFC2234] D. Crocker et P. Overell, "BNF augmenté pour les spécifications de syntaxe : ABNF", novembre 1997.

(*Obsolète, voir* RFC5234)

[RFC2277] H. Alvestrand, "Politique de l'IETF en matière de jeux de caractères et de langages", BCP 18, janvier 1998.

[RFC2279] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", janvier 1998. (*Obsolète, voir* RFC3629) *(D.S.)*

[RFC2389] P. Hethmon, R. Elz, "Mécanisme de négociation de caractéristiques pour FTP", août 1998. *(P.S.)*

[RFC3659] P. Hethmon, "Extensions à FTP", mars 2007. (*MàJ* RFC0959) *(P.S.)*

[UNICODE] The Unicode Consortium, "The Unicode Standard - Version 2.0", Addison Westley Developers Press, juillet 1996.

[UTF-8] ISO/IEC 10646-1:1993 Amendement 2 (1996). “UCS Transformation Format 8 (UTF-8)”.

9. Adresse de l’auteur

Bill Curtin
JIEO
Attn: JEBBD
Ft. Monmouth, N.J. 07703-5613
mél : curtinw@ftm.disa.mil

Annexe A Considérations de mise en œuvre

A.1 Considérations générales

- Les mises en œuvre devraient s’assurer que leur code tient compte des problèmes potentiels, tels que l’utilisation d’un caractère NUL pour terminer une chaîne ou de n’être plus capable de supprimer le bit de poids fort en usage interne, lors de la prise en charge du jeu de caractères étendu.

- Les mises en œuvre devraient savoir qu’il y a des chances pour que les noms de chemin qui ne sont pas en UTF-8 puissent être analysés comme de l’UTF-8 valide. La probabilité est faible pour certains codages ou statistiquement de zéro à zéro pour d’autres. Une analyse non scientifique récente a trouvé que des mots japonais codés en EUC avaient 2,7 % de fausse lecture ; SJIS a 0,0005 % de fausse lecture ; d’autres codages tels que l’ASCII ou le KOI-8 ont 0 % de fausse lecture. Cette probabilité est la plus forte pour les noms de chemin courts et diminue lorsque la taille du nom de chemin augmente. Les mises en œuvre peuvent vouloir chercher des signes que les noms de chemin qui s’analysent comme de l’UTF-8 ne sont pas de l’UTF-8 valide, comme l’existence de plusieurs jeux de caractères locaux dans les noms de chemin courts. Heureusement, comme de plus en plus de mises en œuvre se conforment au codage de transfert UTF-8, il y a de moins en moins besoin de faire des devinettes sur le codage.

- Les développeurs de client devraient être conscients qu’il est possible que les noms de chemin contiennent des caractères mêlés (par exemple, //Nom de répertoire en Latin1/nom de fichier en hébreu). Ils devraient être prêts à traiter l’affichage bidirectionnel (BIDI) de ces jeux de caractères (c’est-à-dire, affichage de droite à gauche pour le répertoire et affichage de gauche à droite pour le nom de fichier). Bien que l’affichage bidirectionnel sorte du domaine d’application du présent document et soit plus compliqué que l’exemple ci-dessus, un algorithme pour l’affichage bidirectionnel se trouve dans la norme UNICODE 2.0 [UNICODE]. Noter aussi que les noms de chemin peuvent avoir un ordre des octets différent et donc avoir une logique et un affichage équivalent dû à l’insertion de caractères de contrôle BIDI à différents points durant la composition. Noter aussi que des jeux de caractères mixtes peuvent aussi poser des problèmes avec les changements de fonte.

- Un serveur qui copie de façon transparente les noms de chemin à partir d’un système de fichiers local peut continuer de le faire. Il appartient alors aux créateurs du fichier local d’utiliser les noms de chemin UTF-8.

- Les serveurs peuvent prendre en charge l’étiquetage de jeu de caractères des fichiers et/ou des répertoires, de façon que des noms de chemin différents puissent avoir des jeux de caractères différents. Le serveur devrait tenter de convertir tous les noms de chemins en UTF-8, mais si il ne peut le faire, il devrait alors laisser ce nom dans sa forme brute.

- Le système d’exploitation de certains serveurs ne rend pas obligatoire les jeux de caractères, mais permet aux

administrateurs de les configurer dans le serveur FTP. Ces serveurs devraient être configurés à utiliser un tableau de transposition particulier (externe ou incorporé). Cela donne la souplesse pour définir différents jeux de caractères pour les différents répertoires.

- Si le système d'exploitation du serveur ne rend pas obligatoire le jeu de caractères et si le serveur FTP ne peut être configuré, le serveur devrait simplement utiliser les octets bruts dans le nom de fichier. Cela peut être ASCII ou UTF-8.

- Si le serveur est un miroir, et veut juste ressembler au site qu'il reflète, il devrait mémoriser exactement les octets du nom de fichier qu'il a reçu du serveur principal.

A.2 Considérations de transition

- Les serveurs qui prennent en charge la présente spécification, auxquels un ancien client (qui ne prend pas en charge la présente spécification) présente un nom de chemin, peuvent presque toujours dire si le nom de chemin est en UTF-8 (voir en B.1) ou dans un autre jeu de code. Afin de prendre en charge ces anciens clients, les serveurs peuvent souhaiter revenir par défaut à un jeu de code non UTF-8. Cependant, comment un serveur prend en charge le non UTF-8 sort du domaine d'application de la présente spécification.

- Les clients qui prennent en charge la présente spécification seront capables de déterminer si le serveur peut accepter l'UTF-8 (c'est-à-dire, prendre en charge la présente spécification) par la capacité du serveur à prendre en charge la commande FEAT et les caractéristiques d'UTF-8 (définies en 3.2). Si des clients plus récents déterminent que le serveur ne prend pas en charge l'UTF-8, il peut souhaiter revenir par défaut à un jeu de code différent. Les développeurs de clients devraient prendre en considération que les noms de chemin, associés à d'anciens serveurs, peuvent être mémorisés en UTF-8. Cependant, comment un client prend en charge le non UTF-8 sort du domaine d'application de la présente spécification.

- Les clients et les serveurs peuvent faire transition vers l'UTF-8 soit en convertissant de/vers le codage local, soit les utilisateurs peuvent mémoriser les noms de fichier en UTF-8. La première approche est plus facile sur les systèmes de fichiers bien contrôlés (par exemple, les PC et les MAC). La dernière approche est plus facile sur des systèmes de fichier de forme plus libre (par exemple, Unix).

- Pour une utilisation interactive, l'attention devrait se concentrer sur l'interface d'utilisateur et la facilité d'utilisation. Une utilisation non interactive exige un comportement cohérent et contrôlé.

- Il peut y avoir de nombreuses applications qui référencent les fichiers sous leur vieux nom de chemin brut (par exemple des liens d'URL). Changer les noms de chemins en UTF-8 va causer l'échec de l'accès aux vieux URL. Une solution peut être que le serveur agisse comme si il y avait deux noms de chemin différents associés au fichier. Cela peut être fait en interne au serveur sur les systèmes de fichiers contrôlés ou en utilisant des liaisons symboliques sur les systèmes de forme libre. Bien que cette approche puisse fonctionner pour une utilisation non interactive d'un seul transfert de fichier, un transfert non interactif de tous les fichiers d'un répertoire va produire des duplications. Les utilisateurs interactifs peuvent se voir présenter des listes de fichiers qui sont des doubles des numéros de fichiers réels.

Annexe B Échantillon de code et exemples

B.1 Vérification de la validité comme UTF-8

Le sous-programme suivant vérifie si une séquence d'octets est de l'UTF-8 valide. Il fait cela en vérifiant le bon étiquetage du premier octet et des suivants pour s'assurer qu'il se conforment au format UTF-8. Il s'assurer ensuite que la partie données de la séquence UTF-8 se conforme à la bonne gamme allouée par le codage. Noter que ce sous programme ne va pas détecter les caractères qui n'ont pas été alloués et n'existent donc pas.

```
int utf8_valid(const unsigned char *buf, unsigned int len)
{
 const unsigned char *endbuf = buf + len;
 unsigned char byte2mask=0x00, c;
 int trailing = 0;                          // des octets de suite (de continuation) à suivre

 while (buf != endbuf)
 {
  c = *buf++;
  if (trailing)
```

```
  if ((c&0xC0) == 0x80)              // Y a-t-il des octets de suite après le format UTF-8 ?
  {if (byte2mask)                    // Faut-il vérifier si le deuxième octet est dans la bonne gamme ?
    if (c&byte2mask)                 // Les bits appropriés sont-ils établis ?
     byte2mask=0x00;
    else
     return 0;
   trailing--; }
  else
   return 0;
 else
  if ((c&0x80) == 0x00)  continue;   // L'octet 1 est de l'UTF-8 valide
  else if ((c&0xE0) == 0xC0)         // L'octet 2 est de l'UTF-8 valide
       if (c&0x1E)                   // L'octet UTF-8 est il dans la bonne gamme ?
        trailing =1;
       else
        return 0;
  else if ((c&0xF0) == 0xE0)         // L'octet 3 est de l'UTF-8 valide
       {if (!(c&0x0F))               // L'octet UTF-8 est il dans la bonne gamme ?
         byte2mask=0x20;             // Si non, établir le gabarit pour vérifier l'octet suivant
         trailing = 2;}
  else if ((c&0xF8) == 0xF0)         // L'octet 4 est de l'UTF-8 valide
       {if (!(c&0x07))               // L'octet UTF-8 est il dans la bonne gamme ?
         byte2mask=0x30;             // Si non, établir le gabarit pour vérifier l'octet suivant
         trailing = 3;}
  else if ((c&0xFC) == 0xF8)         // L'octet 5 est de l'UTF-8 valide
       {if (!(c&0x03))               // L'octet UTF-8 est il dans la bonne gamme ?
         byte2mask=0x38;             // Si non, établir le gabarit pour vérifier l'octet suivant
         trailing = 4;}
  else if ((c&0xFE) == 0xFC)         // L'octet 6 est de l'UTF-8 valide
       {if (!(c&0x01))               // L'octet UTF-8 est il dans la bonne gamme ?
         byte2mask=0x3C;             // Si non, établir le gabarit pour vérifier l'octet suivant
         trailing = 5;}
  else  return 0;
 }
 return trailing == 0;
}
```

B.2 Conversions

Les exemples de code de ce paragraphe reflètent de près l'algorithme de la norme ISO 10646 et peuvent ne pas présenter la solution la plus efficace pour convertir de/vers le codage UTF-8. Si l'efficacité fait problème, les mises en œuvre devraient utiliser les opérateurs au bit près appropriés.

On trouvera des exemples de code supplémentaires et de nombreux tableaux de transposition sur le site de Unicode : http://www.unicode.org ou ftp://unicode.org

Noter que les exemples de conversion ci-dessous supposent que le jeu de caractères local pris en charge dans le système d'exploitation est parfois autre que de l'UCS2/UTF-16. Il y a des systèmes d'exploitation qui prennent déjà en charge l'UCS2/UTF-16 (en particulier Plan 9 et Windows NT). Dans ce cas, aucune conversion ne sera nécessaire du jeu de caractère local à l'UCS.

B.2.1 Conversion de jeu de caractères local en UTF-8

La conversion du jeu de caractères du système de fichiers local à l'UTF-8 va normalement impliquer un processus en deux étapes. D'abord, convertir le jeu de caractères local en UCS, puis convertir l'UCS en UTF-8.

La première étape du processus peut être effectuée en tenant un tableau de transposition qui comporte le code du jeu de caractère local et le code UCS correspondant. Par exemple, le code ISO/CEI 8859-8 [ISO-8859] pour la lettre hébreu "VAV" est 0xE4. Le code correspondant de l'ISO/CEI 10646 sur quatre octets est 0x000005D5.

L'étape suivante est de convertir le code du caractère UCS en codage UTF-8. Le sous-programme suivant peut être utilisé pour déterminer et coder le nombre correct d'octets sur la base du code de caractère UCS-4:

```
unsigned int ucs4_to_utf8 (unsigned long *ucs4_buf, unsigned int ucs4_len, unsigned char *utf8_buf)

{
const unsigned long *ucs4_endbuf = ucs4_buf + ucs4_len;
   unsigned int utf8_len = 0;                  // Retourne la valeur pour la taille UTF8
   unsigned char *t_utf8_buf = utf8_buf;   // Pointeur temporaire pour charger les valeurs UTF8

 while (ucs4_buf != ucs4_endbuf)
 {
 if ( *ucs4_buf <= 0x7F)                       // Caractères ASCII, pas de conversion nécessaire
 {
 *t_utf8_buf++ = (unsigned char) *ucs4_buf;
 utf8_len++;
 ucs4_buf++;
 }
 else
 if ( *ucs4_buf <= 0x07FF )                    // Dans la gamme utf-8 à deux octets
 {
  *t_utf8_buf++= (unsigned char) (0xC0 + (*ucs4_buf/0x40));
  *t_utf8_buf++= (unsigned char) (0x80 + (*ucs4_buf%0x40));
  utf8_len+=2;
  ucs4_buf++;
 }
 else
  if ( *ucs4_buf <= 0xFFFF )                   /* Dans la gamme utf-8 à trois octets. Les valeurs 0x0000FFFE, 0x0000FFFF et
                                                  0x0000D800 - 0x0000DFFF ne se produisent pas en UCS-4 */
  {
   *t_utf8_buf++= (unsigned char) (0xE0 + (*ucs4_buf/0x1000));
   *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x40)%0x40));
   *t_utf8_buf++= (unsigned char) (0x80 + (*ucs4_buf%0x40));
   utf8_len+=3;
   ucs4_buf++;
   }
  else
   if ( *ucs4_buf <= 0x1FFFFF )                // Dans la gamme utf-8 à quatre octets
   {
    *t_utf8_buf++= (unsigned char) (0xF0 + (*ucs4_buf/0x040000));
    *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x10000)%0x40));
    *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x40)%0x40));
    *t_utf8_buf++= (unsigned char) (0x80 + (*ucs4_buf%0x40));
    utf8_len+=4;
    ucs4_buf++;

   }
   else
    if ( *ucs4_buf <= 0x03FFFFFF )           // Dans la gamme utf-8 à cinq octets
    {
     *t_utf8_buf++= (unsigned char) (0xF8 + (*ucs4_buf/0x01000000));
     *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x040000)%0x40));
     *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x1000)%0x40));
     *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x40)%0x40));
     *t_utf8_buf++= (unsigned char) (0x80 + (*ucs4_buf%0x40));
     utf8_len+=5;
     ucs4_buf++;
    }
    else
    if ( *ucs4_buf <= 0x7FFFFFFF )           // Dans la gamme utf-8 à six octets
     {
       *t_utf8_buf++= (unsigned char) (0xF8 +(*ucs4_buf/0x40000000));
       *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x01000000)%0x40));
       *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x040000)%0x40));
       *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x1000)%0x40));
```

```
        *t_utf8_buf++= (unsigned char) (0x80 + ((*ucs4_buf/0x40)%0x40));
        *t_utf8_buf++= (unsigned char) (0x80 + (*ucs4_buf%0x40));
        utf8_len+=6;
        ucs4_buf++;
       }
 }
 return (utf8_len);
}
```

B.2.2 Conversion de UTF-8 en jeu de caractère local

Quand on passe du codage UTF-8 au jeu de caractères local, on utilise la procédure inverse. D'abord, le codage UTF-8 est transformé en jeu de caractères UCS-4. L'UCS-4 est ensuite converti dans le jeu de caractères local à partir d'un tableau de transposition (c'est-à-dire l'opposé du tableau utilisé pour former le code de caractères UCS-4).

Pour convertir de l'UTF-8 en UCS-4, les bits libres (ceux qui ne définissent pas la taille de la séquence UTF-8 ou qui signifient des octets de continuation) dans une séquence UTF-8 sont enchaînés comme une chaîne binaire. Les bits sont alors répartis dans une séquence de quatre octets en commençant pas les bits de moindre poids. Les bits qui ne sont pas alloués à un bit de la séquence de quatre octets sont bourrés de bits à zéro. Le sous programme suivant convertit le codage UTF-8 en codes de caractères UCS-4 :

```
int utf8_to_ucs4 (unsigned long *ucs4_buf, unsigned int utf8_len, unsigned char *utf8_buf)
{

const unsigned char *utf8_endbuf = utf8_buf + utf8_len;
unsigned int ucs_len=0;

 while (utf8_buf != utf8_endbuf)
 {

 if ((*utf8_buf & 0x80) == 0x00)             /*caractères ASCII, pas de conversion nécessaire */
 {
 *ucs4_buf++ = (unsigned long) *utf8_buf;
 utf8_buf++;
 ucs_len++;
 }
 else
 if ((*utf8_buf & 0xE0)== 0xC0)              // Dans la gamme utf-8 à deux octets
 {
  *ucs4_buf++ = (unsigned long) (((*utf8_buf - 0xC0) * 0x40) + ( *(utf8_buf+1) - 0x80));
  utf8_buf += 2;
  ucs_len++;
 }
 else
   if ( (*utf8_buf & 0xF0) == 0xE0 )         //* Dans la gamme utf-8 à trois octets */
   {
   *ucs4_buf++ = (unsigned long) (((*utf8_buf - 0xE0) * 0x1000)
                + (( *(utf8_buf+1) -  0x80) * 0x40)
                + ( *(utf8_buf+2) - 0x80));
    utf8_buf+=3;
    ucs_len++;
   }
   else
    if ((*utf8_buf & 0xF8) == 0xF0)          /* Dans la gamme utf-8 à quatre octets */
    {
     *ucs4_buf++ = (unsigned long)
                  (((*utf8_buf - 0xF0) * 0x040000)
                  + (( *(utf8_buf+1) -  0x80) * 0x1000)
                  + (( *(utf8_buf+2) -  0x80) * 0x40)
                  + ( *(utf8_buf+3) - 0x80));
     utf8_buf+=4;
     ucs_len++;
```

```
    }
    else
     if ((*utf8_buf & 0xFC) == 0xF8)          /* Dans la gamme utf-8 à cinq octets */
     {
      *ucs4_buf++ = (unsigned long)
                  (((*utf8_buf - 0xF8) * 0x01000000)
                  + ((*(utf8_buf+1) - 0x80) * 0x040000)
                  + (( *(utf8_buf+2) -  0x80) * 0x1000)
                  + (( *(utf8_buf+3) -  0x80) * 0x40)
                  + ( *(utf8_buf+4) - 0x80));
      utf8_buf+=5;
      ucs_len++;
     }
     else
      if ((*utf8_buf & 0xFE) == 0xFC)         /* Dans la gamme utf-8 à six octets */
      {
       *ucs4_buf++ = (unsigned long)
                   (((*utf8_buf - 0xFC) * 0x40000000)
                    + ((*(utf8_buf+1) - 0x80) * 0x010000000)
                    + ((*(utf8_buf+2) - 0x80) * 0x040000)
                    + (( *(utf8_buf+3) -  0x80) * 0x1000)
                    + (( *(utf8_buf+4) -  0x80) * 0x40)
                    + ( *(utf8_buf+5) - 0x80));
       utf8_buf+=6;
       ucs_len++;
      }

 }
return (ucs_len);
}
```

B.2.3 Exemple de ISO/CEI 8859-8

Cet exemple montre la transposition du jeu de caractères ISO/CEI 8859-8 en UTF-8 et retour à l'ISO/CEI 8859-8. Comme noté précédemment, la lettre hébreu "VAV" est convertie du code de caractère ISO/CEI 8859-8 0xE4 au code correspondant de l'ISO/CEI 10646 à quatre octets 0x000005D5 par une simple recherche dans un fichier de conversion/transposition.

Le code du caractère UCS-4 est transformé en UTF-8 en utilisant le sous-programme ucs4_to_utf8 décrit plus haut par :
1. Comme le caractère UCS-4 est entre 0x80 et 0x07FF, il sera transposé en une séquence UTF-8 de deux octets.
2. Le premier octet est défini par (0xC0 + (0x000005D5 / 0x40)) = 0xD7.
3. Le second octet est défini par (0x80 + (0x000005D5 % 0x40)) = 0x95.

Le codage UTF-8 est retransféré en UCS-4 en utilisant le sous-programme utf8_to_ucs4 décrit plus haut par :
1. Comme le premier octet de la séquence, lorsque l'opérateur '&' est appliqué avec une valeur de 0xE0, va produire 0xC0 (0xD7 & 0xE0 = 0xC0) l'UTF-8 est une séquence de deux octets.
2. Le code UCS-4 à quatre octets est produit par (((0xD7 - 0xC0) * 0x40) + (0x95 -0x80)) = 0x000005D5.

Finalement, le code de caractère UCS-4 est converti en code de caractère ISO/CEI 8859-8 (en utilisant le tableau de transposition qui correspond à l'ISO/CEI 8859-8 en UCS-4) pour produire le code original 0xE4 pour la lettre hébreu "VAV".

B.2.4 Exemple de page de code de fabricant

Cet exemple montre la transposition d'une page de code en UTF-8 et retour à une page de code de fabricant. La transposition entre les pages de code de fabricant peut être faite d'une manière très similaire à celle décrite ci-dessus. Par exemple les deux pages de code de PC et de Mac reflètent le jeu de caractères de la norme thaï TIS 620-2533. Le code de caractère sur les deux plateformes pour la lettre thaï "SO SO" est 0xAB. Ce caractère peut alors être transposé en UCS-4 au moyen d'un fichier de conversion/transposition pour produire le code UCS-4 de 0x0E0B.

Le code de caractère UCS-4 est transformé en UTF-8 en utilisant le sous-programme ucs4_to_utf8 décrit plus haut par :
1. Comme le caractère UCS-4 est entre 0x0800 et 0xFFFF il va se transposer en une séquence UTF-8 de trois octets.
2. Le premier octet est défini par (0xE0 + (0x00000E0B / 0x1000) = 0xE0.

3. Le second octet est défini par (0x80 + ((0x00000E0B / 0x40) % 0x40))) = 0xB8.
4. Le troisième octet est défini par (0x80 + (0x00000E0B % 0x40)) = 0x8B.

Le codage UTF-8 est retransféré en UCS-4 en utilisant le sous-programme utf8_to_ucs4 décrit plus haut par :
1. Comme le premier octet de la séquence, lorsque on applique l'opérateur '&' avec une valeur de 0xF0, va produire 0xE0 (0xE0 & 0xF0 = 0xE0) l'UTF-8 est une séquence de trois octets.
2. Le code de caractère UCS-4 à quatre octets est produit par (((0xE0 - 0xE0) * 0x1000) + ((0xB8 - 0x80) * 0x40) + (0x8B -0x80) = 0x0000E0B.

Finalement, le code de caractère UCS-4 est converti en page de code de caractère soit PC, soit MAC (en utilisant le tableau de transposition qui correspond à la page de code UCS-4) pour produire le code original 0xAB pour la lettre thaï "SO SO".

B.3 Pseudo code pour un serveur à haute qualité de traduction

```
 si utf8_valid(fn)
  {
  tente de convertir fn dans le jeu de caractère local , produisant localfn
  si (échec temporaire de la conversion) retourne une erreur
  si (conversion réussie)
  {
    tente d'ouvrir localfn
    si (échec temporaire de l'ouverture) retourne une erreur
    si (l'ouverture réussit) retourne l'indication de succès
  }
  }
tente d'ouvrir fn
si (échec temporaire d'ouverture) retourne une erreur
si (l'ouverture réussit) retourne une indication de succès
retourne une erreur permanente
```

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'éditeur des RFC est actuellement fourni par la Internet Society.