Chapitre 2

Structure générale

Ce chapitre traite des principes fondamentaux qui régissent la conception du standard Unicode et présente un aperçu de ses principales caractéristiques. Il examine les différents traitements textuels, les principes d'unification, l'affectation des caractères au sein de l'espace de codage, les propriétés des caractères, la direction d'écriture et, enfin, une description des signes combinatoires et de la manière dont ils peuvent être employés avec Unicode. Ce chapitre décrit également les exigences générales relatives à la création de systèmes de manipulation de texte conformes au standard Unicode. Les exigences formelles de conformité sont présentées au chapitre *3, Conformité.* On trouvera les propriétés des caractères, normatives et informatives, au chapitre *4, Propriétés des caractères*. Enfin le chapitre *5, Conseils de mise en œuvre*, expose un ensemble de principes directeurs particulièrement utiles pour ceux qui désirent implanter Unicode.

2.1 Contexte architectural

Un jeu de caractères tel que le standard Unicode permet de mettre en œuvre un ensemble de traitements textuels particulièrement utiles. Il faut se souvenir que les aspects intéressants ne sont pas tellement les codes de caractères, mais bien les processus qui opèrent sur des textes, car c'est eux qui répondent directement aux besoins des utilisateurs. Les codes de caractères sont un peu les rouages –- cachés mais omniprésents et essentiels — utilisés pour construire de mille et une façons les systèmes informatiques. Aucune architecture de jeu de caractères ne pouvant satisfaire de manière optimale tous les usages, l'architecture du standard Unicode est donc un compromis entre des exigences divergentes.

Manipulation de texte de base

La plupart des systèmes informatiques fournissent des fonctionnalités de bas niveau pour un petit nombre de processus textuels de base à partir desquelles on construit des fonctions de manipulation de texte plus poussées. Les processus textuels suivants sont pris en charge, dans une certaine mesure, par la plupart des systèmes informatiques :

- rendu des caractères (y compris les ligatures, les formes contextuelles, etc.) ;
- coupure de lignes lors du rendu, y compris la césure (coupure de mots et insertion de traits d'union) le cas échant ;
- modification de l'apparence : force du corps (la taille en points), crénage, soulignement, inclinaison et graisse (maigre, demi-gras, gras, etc.) ;
- détection d'unités telles que les « mots » ou les « phrases » ;
- interaction avec les utilisateurs pour la sélection ou la mise en évidence d'un élément textuel ;
- modification de la saisie et édition du texte stocké grâce à l'insertion ou à la suppression de caractères ;
- comparaison de textes afin de déterminer l'ordre lexicographique de deux chaînes, de filtrer ou de faire correspondre des chaînes ;

- analyse du contenu du texte pour effectuer des opérations comme la vérification orthographique, la coupure de mots, l'analyse morphologique (c'est-à-dire la détermination du radical d'un mot et de ses affixes) ;
- compression ou décompression, troncature, chiffrement, déchiffrement, émission ou réception de texte en masse.

Éléments textuels, valeurs de codes et traitements textuels

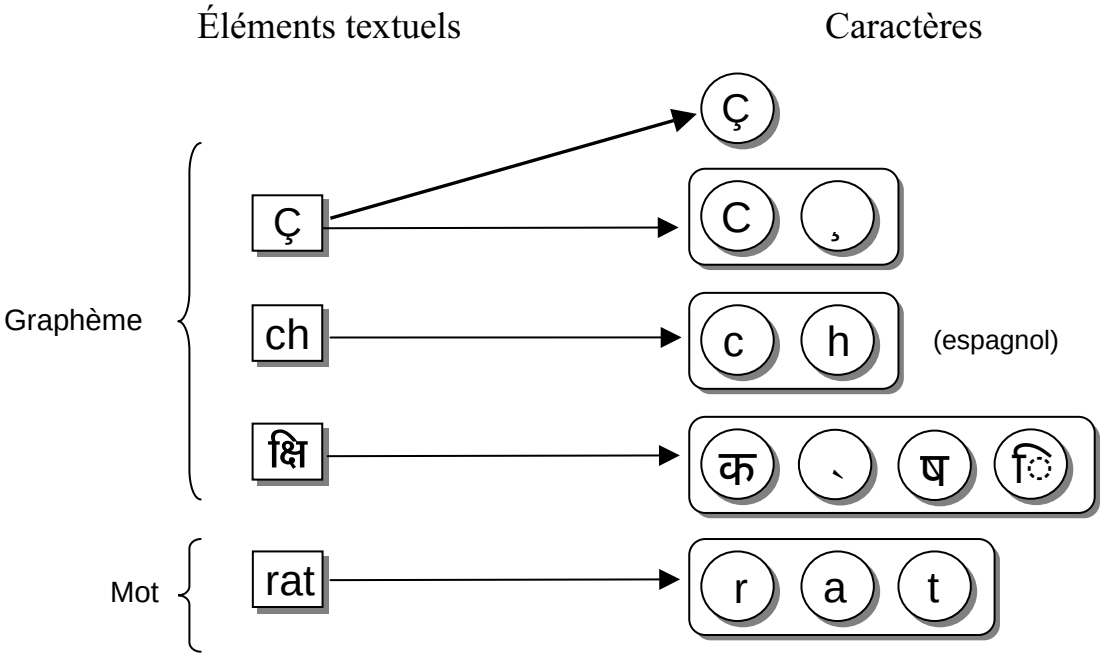
Un des défis les plus redoutables dans la conception d'un codage de caractères universel réside dans le fait que chaque langue écrite a une conception différente, pour chaque processus textuel, de ce qui constitue une unité textuelle fondamentale, également dénommée *élément textuel*.

Ainsi, en typographie traditionnelle allemande, la combinaison de lettres « ck » constitue un élément textuel pour ce qui a trait à la coupure de mots (où cette combinaison se transforme en « k-k ») mais pas en ce qui concerne le tri lexicographique. En espagnol, la combinaison « ll » peut constituer un élément textuel pour le tri lexicographique traditionnel (où il s'insère entre le « l » et le « m ») mais pas dans le cas du rendu. En français, les objets « a » et « A » sont habituellement considérés comme des éléments textuels distincts pour ce qui a trait au rendu, mais on les confond fréquemment à dessein lors de recherches textuelles. Les éléments textuels d'une langue dépendent du processus textuel en question : un élément textuel destiné à une vérification orthographique pourra avoir des limites différentes d'un élément textuel destiné au classement.

Une norme de codage de caractères fournit des unités fondamentales de codage (c'est-à-dire des caractères abstraits) mises en correspondance biunivoque (de un à un) avec les numéros de code attribués. Ces valeurs de codes sont les plus petites unités adressables du texte stocké.

Le graphème constitue une des classes d'éléments textuels importantes, il correspond typiquement à ce que l'utilisateur considère être un caractère. La figure 2-1 illustre le rapport qui existe entre les caractères abstraits et les graphèmes.

Figure 2-1. Éléments textuels et caractères



L'architecture d'un codage de caractères doit fournir un jeu précis de caractères qui permette aux programmeurs de concevoir des applications capables de mettre en œuvre une grande variété de processus textuels dans les langues ciblées.

Traitements textuels et codage

Dans le cas d'un texte anglais qui utilise un mécanisme de codage comme l'ASCII, le lien entre le codage et les processus textuels de base fondés sur ce codage sont en apparence assez simples : les caractères sont généralement affichés un à un de gauche à droite et dans l'ordre linéaire dans des rectangles indépendants. Ainsi, dans un processus comme l'affichage élémentaire de l'anglais, un code de caractère stocké correspond à un caractère logique.

Lors de la conception d'un codage capable de traiter des textes internationaux ou multilingues, tel que le standard Unicode, il faut considérer explicitement le lien entre le codage et la mise en œuvre des processus textuels de base pour plusieurs raisons :

- De nombreuses hypothèses relatives au rendu des caractères qui s'appliquent à l'anglais ne s'appliquent pas aux autres systèmes d'écriture. Contrairement à l'anglais, les caractères appartenant à d'autres systèmes d'écriture ne sont pas nécessairement affichés un à un dans des rectangles rangés de gauche à droite. À maintes reprises, le positionnement des caractères est assez complexe et ne s'effectue pas d'une manière linéaire. Voir les sections *9.2, Arabe* au chapitre *9, Écritures du Moyen-Orient* et *10.1, Dévanâgarî* au chapitre *10, Écritures du Sud et du Sud-Est asiatique* pour plus de détails sur cet aspect.
- Il n'est pas toujours évident qu'un seul jeu de caractères textuels représente le codage optimal pour une langue donnée. Ainsi, on peut envisager deux manières de coder les caractères accentués fréquents en français ou en suédois : l'ISO 8859-1 définit les lettres « ä » et « ö » en tant que caractères séparés, alors que l'ISO 5426 les représente plutôt par composition. En suédois, ces deux lettres sont distinctes et triées à la suite de « z ». Par contre en français, le tréma sur une voyelle indique simplement que la lettre est prononcée de manière isolée. En pratique, on peut utiliser ces deux méthodes pour coder ces deux langues.
- Aucun codage ne peut prendre en charge tous les processus textuels de base de façon optimale. Il faut donc faire quelques compromis. ASCII, par exemple, attribue des codes séparés aux majuscules et aux minuscules. Ce choix permet de simplifier quelques processus textuels, tels que le rendu, mais d'autres processus, comme la comparaison, sont légèrement plus complexes. Une autre architecture de codage pour l'anglais, fondée sur des codes de commande de changement de casse, aurait l'effet inverse. Lors de la conception d'un nouveau mécanisme de codage destiné à des écritures complexes, il faut évaluer ce genre de compromis et prendre des décisions de manière délibérée plutôt qu'inconsciemment.

Pour ces raisons, l'architecture du standard Unicode ne privilégie pas la conception d'algorithmes de manipulation de texte particuliers. Unicode fournit plutôt un codage qui s'adapte à une grande gamme d'algorithmes. Les algorithmes de tri et de comparaison de chaînes, notamment, *ne doivent pas* supposer que les numéros de code de caractères Unicode fournissent un moyen direct de classer les chaînes en un ordre lexicographique quelconque. La mise en œuvre d'un tri lexicographique adapté à une culture particulière peut nécessiter un algorithme d'une complexité arbitraire. L'ordre lexicographique des caractères diffère selon la langue (cf. La section *5.17, Tri et repérage* pour les lignes directrices de mise en œuvre).

Les processus textuels multilingues sont souvent plus compliqués que ceux qui ne prennent en charge que le français. L'architecture de codage de caractères du standard Unicode s'efforce de minimiser cette complexité supplémentaire et permet ainsi aux systèmes informatiques modernes d'échanger, d'afficher et de manipuler du texte dans la langue et l'écriture de l'utilisateur et, peut-être, d'autres langues également.

2.2 Principes de conception

La conception du standard Unicode reflète les 10 principes fondamentaux énoncés au tableau 2-1. Tous ces principes ne peuvent être satisfaits simultanément. La conception cherche à concilier, d'une part, la cohérence à des fins de simplicité et d'efficacité et, d'autre part, la compatibilité afin d'assurer l'échange avec des normes existantes.

Tableau 2-1. Les 10 principes de conception d'Unicode

Principe	Énoncé
Codes de caractère à 16 bits	Les codes de caractère Unicode ont une largeur de 16 bits, ils forment un seizet.
Efficacité	L'analyse et le traitement d'un texte Unicode sont simples.
Caractères et non glyphes	Le standard Unicode code des caractères et non des glyphes.
Sémantique	Les caractères ont une sémantique bien définie.
Texte brut	Le standard Unicode code du texte brut.
Ordre logique	La représentation implicite en mémoire est l'ordre logique.
Unification	Le standard Unicode unifie les caractères identiques à l'intérieur des écritures quelles que soient les langues.
Composition dynamique	On compose les formes accentuées dynamiquement.
Séquence équivalente	Chaque forme statique précomposée a une suite équivalente de caractères dynamiquement composés.
Convertibilité	Une convertibilité exacte est garantie entre le standard Unicode et d'autres normes largement répandues.

Codes de caractère à 16 bits

Dans sa forme UTF-16 (cf. section 2.3), un texte brut Unicode consiste en une suite d'unités de stockage Unicode codées sur 16 bits. Ces seizets sont faciles à analyser et permettent d'efficacement trier, repérer, afficher ou éditer des chaînes textuelles. Parmi les 65 536 valeurs de code du *plan multilingue de base*, 63 486 peuvent être utilisées pour représenter des caractères avec un seul seizet, 2 048 valeurs de code sont réservées pour la représentation de 1 048 544 caractères supplémentaires à l'aide de seizets appariés. Ces valeurs de code appariées, également appelées seizets d'indirection, permettent aux mises en œuvre d'Unicode utilisant UTF-16 d'accéder aux caractères stockés dans les plans complémentaires.

L'ensemble complet des 256 valeurs possibles peut être utilisé par chacun des deux octets qui forment les valeurs de code Unicode à 16 bits. Un processus architecturé autour d'unités codées sur 8 bits, et pour qui certaines de ces valeurs sont réservées, pourrait échouer s'il

devait inopinément rencontrer un seizet Unicode. À des fins de compatibilité avec les environnements en service, il existe une transformation sans perte qui exprime les caractères Unicode en un format approprié pour les environnements à 8 bits, il s'agit d'UTF-8.

L'UTF-8 (format transformé d'Unicode sur 8 bits) est une méthode standard de transformation des numéros de caractère en une suite de codes à 8 bits. L'UTF-8 est destiné à être utilisé quand il est nécessaire d'utiliser des codes à 8 bits ou quand l'intervalle des valeurs ASCII doit être conservé, par exemple lors de la transmission de données au moyen d'un protocole sur 8 bits ou de l'appel de fonctions prévues pour 8 bits.

Pour plus de renseignements sur les formats de transformation d'Unicode, veuillez vous référer à la section *2.3, Formes de codage* et à l'annexe *C.3, Formats de transformation JUC* de ce livre. Veuillez également consulter les formats de transformation de l'ISO/CEI 10646. Pour un format adapté aux systèmes EDBCIC, référez-vous au rapport technique Unicode n° 16, UTF-EBCDIC sur le disque accompagnant ce livre ou sur le site Internet Unicode pour une version tenue à jour.

Efficacité

L'objectif du standard Unicode est de permettre des mises en œuvre efficaces. Unicode n'inclut pas de caractères d'échappement ou d'états de passage. Tous les caractères Unicode ont la même qualité, tous les codes sont d'une égale facilité d'accès.

Toutes les formes de codage sont autosynchronisantes. Quand on accède aléatoirement une chaîne de caractères, il suffit de reculer d'un nombre limité d'octets pour retrouver le début d'un caractère. Dans le cas d'UTF-16, si un pointeur pointe vers le seizet d'indirection inférieur, il suffit de reculer d'une position. Pour UTF-8, si le pointeur pointe vers un octet qui commence par 10xxxxxx (en binaire), il suffit de reculer d'une à trois positions pour trouver le début du caractère.

Par convention et dans la mesure du possible, les caractères d'une même écriture sont regroupés. Non seulement ceci facilite-t-il la consultation des tableaux de caractères, mais cela permet également des mises en œuvre plus compactes. Les caractères de ponctuation communs aux différentes écritures sont regroupés.

Des caractères et non des glyphes

Le standard Unicode distingue les caractères, qui forment les plus petites unités distinctives et significatives d'une langue écrite, des glyphes qui représentent les différentes formes qu'un caractère peut prendre. Différents scénarios existent : un seul glyphe peut représenter un seul ou plusieurs caractères ou de multiples glyphes peuvent représenter un seul caractère. La figure 2-2 illustre les différences entre caractère et glyphe.

Les caractères Unicode représentent principalement, mais pas exclusivement, les lettres, la ponctuation et les autres signes que l'on retrouve dans les notations techniques ou les textes en langue naturelle. On représente un caractère par un numéro qui ne réside qu'en mémoire ou sur disque. Le standard Unicode porte uniquement sur les codes de caractères et non les glyphes.

Contrairement aux caractères, les glyphes apparaissent à l'écran ou sur papier et représentent une forme particulière d'un ou plusieurs caractères. Un répertoire de glyphes constitue une police. La forme des glyphes et les méthodes d'identification et de sélection de glyphes sont la

responsabilité des fonderies et des normes appropriées. Celles-ci ne font pas partie du standard Unicode.

Figure 2-2. Caractères et glyphes

Glyphes (œils)	Caractères Unicode
A 𝐀 𝔸 A A 𝘈 𝔄 A	U+0041 LETTRE MAJUSCULE LATINE A
a 𝐚 ɑ a a 𝒶 𝔞 a	U+0061 LETTRE MINUSCULE LATINE A
fi fi	U+0066 LETTRE MINUSCULE LATINE F + U+0069 LETTRE MINUSCULE LATINE I
ه ٨ ﻫ ﻬ	U+0647 LETTRE ARABE HÉ'

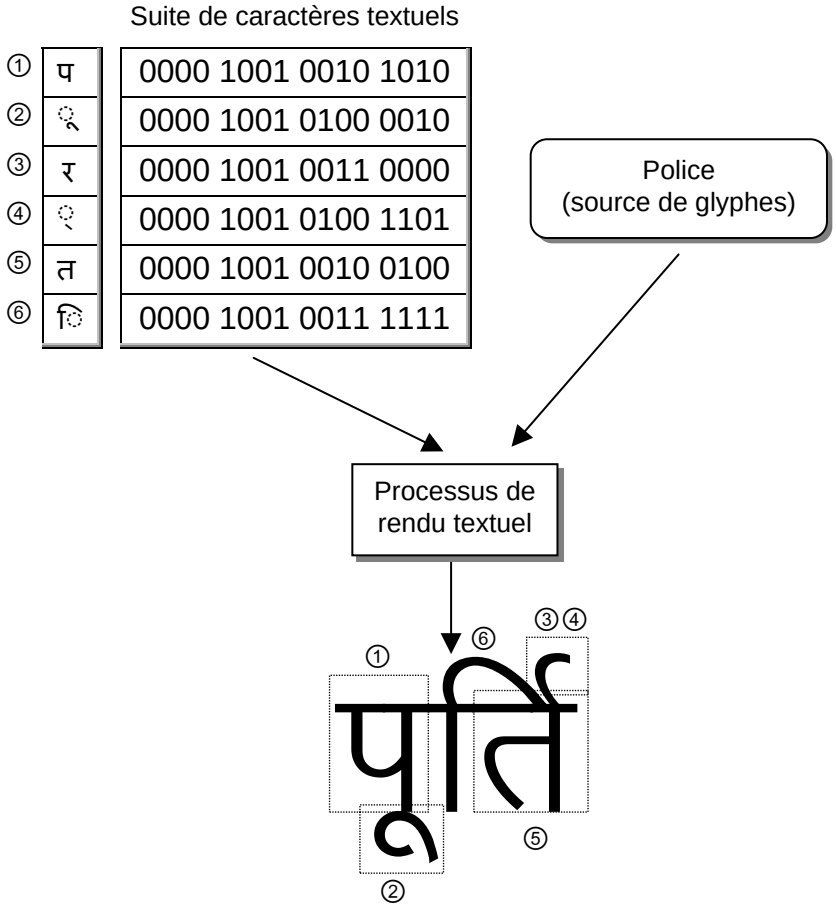
Pour certaines écritures, comme l'arabe et différentes écritures du sous-continent indien, le nombre de glyphes nécessaires à l'affichage peut être nettement plus important que le nombre des caractères qui codent les unités de base de cette écriture. Le nombre de glyphes peut également dépendre du style calligraphique de la police. Ainsi, une police arabe de style nastaliq peut comprendre plusieurs milliers de glyphes. Cependant, le codage de caractères utilise la même cinquantaine de lettres quel que soit le style de police utilisé pour illustrer en contexte les données de caractères.

Une police et le processus de rendu qui lui est associé définissent une correspondance arbitraire entre des valeurs Unicode et des glyphes. Certains glyphes dans une police peuvent être des formes indépendantes de caractères particuliers alors que d'autres représentent des formes qui ne correspondent directement à aucun caractère isolé.

La mise en correspondance entre les caractères stockés en mémoire et les glyphes ne constitue qu'une partie du rendu de texte. L'aspect final du texte rendu peut également dépendre du contexte (les caractères contigus dans la représentation en mémoire), des variations dans le dessin des polices utilisées ainsi que de paramètres de formatage (force du corps, exposant, indice et ainsi de suite). Comme l'illustre la figure 2-3, le résultat à l'écran ou sur papier peut être très différent des formes modèles d'une lettre ou d'un caractère.

Il existe pour toutes les écritures une relation archétypique entre les suites de codes de caractère et l'image glyphique qui en résulte. Pour les écritures dites latines, la relation est simple et notoire ; pour de nombreuses autres écritures, moins connues, cet ouvrage explique cette relation. Cependant, dans tous les cas, une typographie fine peut nécessiter un ensemble de règles plus détaillées que celles précisées ici. Le standard Unicode documente la relation implicite qui existe entre les suites de caractères et l'image glyphique dans la seule intention qu'un même contenu textuel soit toujours stocké à l'aide d'une suite de codes de caractère identique et donc échangeable.

Figure 2-3. Des codes de caractère aux glyphes rendus



Sémantique

Les caractères sont munis d'une sémantique bien définie. Le standard Unicode fournit des tableaux de propriétés de caractère que l'on peut utiliser pour le tri, l'analyse ou d'autres algorithmes qui ont besoin d'une connaissance sémantique des caractères traités. Cf. les sections *5.15, Repérage des frontières d'élément textuel*, *5.16, Identificateurs* et *5.17, Tri et repérage* pour des conseils de mise en œuvre. Les propriétés reprises par le standard Unicode comprennent les propriétés numérales, combinatoires, d'espacement et de directionalité (cf. chapitre *4, Propriétés des caractères*). Au besoin, de temps en temps, il est possible de définir des propriétés supplémentaires. Ni le nom du caractère ni son numéro de code ne permettent de déduire directement ses propriétés. Pour une exception, cf. la section *4.1, Casse – normatif.*

Texte brut

Un *texte brut* est constitué d'une pure suite de codes de caractères, dont aucun ne représente du balisage. Un texte brut Unicode est donc une suite simple de codes de caractères Unicode. À l'inverse, un *texte enrichi*, également connu sous le nom de *texte de fantaisie*, est constitué de texte brut et d'un ensemble d'informations comme l'identification de la langue du texte, la force du corps, la couleur du texte, des hyperliens, etc. Le texte de ce livre, par exemple, est un texte à plusieurs polices formaté à l'aide d'un logiciel d'éditique, il s'agit d'un texte enrichi.

Un texte de fantaisie peut inclure de nombreux types de structures de données. Ainsi, un texte de fantaisie idéographique pourra contenir une transcription phonétique des idéogrammes.

La simplicité du texte brut lui permet de servir d'élément structurant de base pour les textes de fantaisie. SGML, HTML, XML ou TEX sont des exemples de textes de fantaisie représentés à l'aide d'un texte brut émaillé de suite de caractères qui représentent des structures de données supplémentaires (c.-à-d. des balises). La plupart des progiciels de traitement de texte répandus utilisent un tampon de texte brut pour représenter le contenu du document et le relient à une zone parallèle de stockage de données de formatage.

Les textes brut et de fantaisie ont des fonctions bien établies :

- Le texte brut constitue le contenu sous-jacent que l'on peut formater ;
- Le texte brut est public, normalisé et lisible par tous ;
- La représentation du texte de fantaisie peut être propriétaire ou varier selon la mise en œuvre.

Bien qu'on ait normalisé ou rendu publics certains formats de texte de fantaisie, la grande majorité des spécifications de texte de fantaisie sert de vecteurs à des mises en œuvre particulières et ne sont pas nécessairement lisibles par d'autres programmes. Étant donné qu'un texte de fantaisie est formé d'un texte brut et d'informations supplémentaires, il est toujours possible d'éliminer ces informations supplémentaires pour révéler le texte « pur » sous-jacent. On emploie souvent cette opération, par exemple, dans les traitements de texte qui utilisent à la fois leur format privé de fantaisie et un format de texte brut comme moyens d'échange universels, même s'ils sont quelque peu limités. On peut donc dire que le texte brut représente, implicitement, le contenu textuel de base échangeable.

Les normes de langages balisés, comme XML et HTML, utilisent du texte pour l'ensemble du fichier codé. Elles utilisent des conventions spéciales imbriquées au sein du texte tel que « <p> » pour distinguer la balise du contenu vraiment textuel. XML, notamment, utilise Unicode comme codage de base : « Tous les processeurs XML doivent être à même de lire les entités en UTF-8 ou UTF-16 » (recommandation W3C : Langage de balisage extensible (XML) 1.0, § 4.3.3). Les entités XML qui n'utilisent pas ces jeux de caractères doivent explicitement déclarer le codage utilisé.

Étant donné que le texte brut ne fait qu'énumérer les caractères contenus dans ce texte, il n'a pas d'apparence inhérente. Pour le rendre visible, il faut lui associer un processus de rendu. Si le même texte brut devait être rendu par deux processus différents, rien ne garantit que les deux processus produisent des textes d'apparence identique. Au contraire, il suffit simplement que les deux processus de rendu produisent des textes lisibles pour l'écriture ou les écritures impliquées. On peut donc résumer le rapport qui existe entre l'apparence et le contenu d'un texte brut de la façon suivante :

> *Le texte brut doit comprendre suffisamment d'information pour que le texte soit rendu de manière lisible et rien d'autre.*

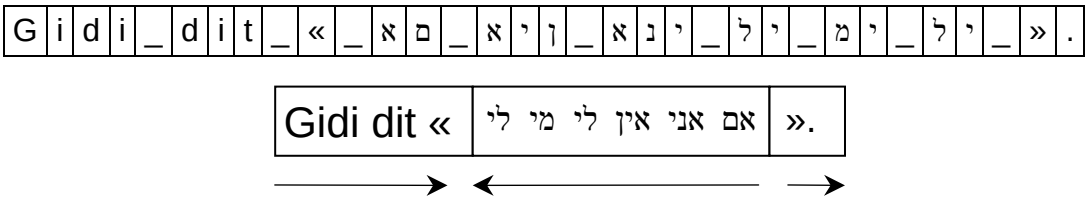
Le standard Unicode code le texte brut. Il revient à un protocole de niveau supérieur de faire la distinction entre les données codées par le standard Unicode et d'autres formes de données présentes dans un même flux de données. Le standard Unicode ne définit pas de méthode qui permet de faire cette distinction. Unicode a conservé les 64 positions de code de commande de l'ISO/CEI 6429 (fréquemment utilisées avec l'ISO/CEI 646 et l'ISO/CEI 8859) pour des

raisons de compatibilité ; elles peuvent être utilisées pour mettre en œuvre de tels protocoles de niveau supérieur (cf. la section *2.8, Commandes et suite de commandes*).

Ordre logique

Les textes Unicode, quelle que soit l'écriture, sont stockés en mémoire en ordre logique. Cet ordre correspond *grosso modo* à l'ordre dans lequel le texte est saisi au clavier. Parfois, l'ordre des caractères à l'affichage ou à l'impression diffère de l'ordre logique. Quand il nécessaire d'assurer une lisibilité constante, le standard Unicode définit la conversion du texte Unicode à partir de sa représentation en mémoire en un texte lisible (affiché). La figure 2-4 illustre la différence entre les ordres logique et d'affichage.

Figure 2-4. Ordre bidirectionnel



À l'affichage, le glyphe qui représente le premier caractère du texte français de la figure 2-4 apparaît à la gauche. Cependant, le premier caractère logique du texte hébreu est représenté par le glyphe hébreu le plus proche de la marge droite. Les glyphes hébreux suivants sont disposés vers la gauche.

L'ordre logique est de mise même lorsque des caractères de différentes directions dominantes se mêlent : écritures gauche-à-droite (grec, cyrillique, latin), droite-à-gauche (arabe, hébreu) ou même verticale. Les propriétés de directionalité implicites aux caractères suffisent généralement à déterminer l'ordre correct d'affichage. Il arrive parfois que cette directionalité implicite s'avère insuffisante pour afficher le texte de manière lisible. Cette situation se produit quand on mélange des écritures de différentes directionalités. C'est pourquoi le standard Unicode comprend des caractères qui permettent de modifier la direction implicite. Le chapitre *3, Conformité* stipule un ensemble de règles pour la représentation correcte des textes dont certains segments sont écrits de gauche à droite et d'autres de droite à gauche. On qualifie ces textes de bidirectionnels.

Dans l'ensemble, l'ordre logique correspond à l'*ordre phonétique*. Les seules exceptions actuelles sont les écritures thaïe et lao qui utilisent l'ordre visuel, car les scripteurs pour ces langues saisissent les caractères dans l'ordre visuel plutôt que dans l'ordre phonétique.

Certains caractères, comme l'*i bref* dévanâgarî, s'affichent avant les caractères qu'ils suivent logiquement en mémoire (cf. la section *10.1, Dévanâgarî*).

Les signes combinatoires (comme les accents en français, en grec, en cyrillique, les points voyelles arabes, les signes voyelles dévanâgarî, etc.) n'apparaissent pas de manière linéaire dans le texte rendu. Dans une chaîne de caractères Unicode, tous ces signes *suivent* le caractère de base qu'ils modifient. Ainsi, un « ã » latin est-il stocké sous la forme d'un « a » suivi du signe combinatoire (ou diacritique) « ◌̃ » quand on n'utilise pas la forme précomposée de ce caractère.

Unification

Le standard Unicode évite de coder plusieurs fois le même caractère en affectant un seul numéro de code aux caractères utilisés par plusieurs langues au sein d'une même écriture. Chaque lettre commune, signe de ponctuation, symbole et diacritique se voit attribuer un seul code, quelle que soit la langue. Il en est de même de certains idéophonogrammes chinois/japonais/coréens (CJC) (cf. section *11.1, han*).

Il faut prendre soin de ne pas effectuer de distinctions artificielles parmi les caractères. Il peut être déroutant pour un utilisateur de voir sur son écran un *Å* sans pourtant pouvoir le retrouver dans le texte à l'aide du dialogue de recherche. Ceci pourrait se produire car l'utilisateur ne voit pas un *Å* (A rond) mais un *Å* (Angstrœm ou Angström). On appelle ce phénomène l'ambigüiïté visuelle.

Il est tout à fait normal que de nombreux caractères jouent plusieurs rôles. Ainsi la *virgule* « , » sert-elle de séparateur décimal en français alors qu'en typographie américaine il s'agit du séparateur des milliers. Le standard Unicode ne dédouble pas les caractères pour des raisons d'utilisation variable d'après la langue, mais il le fait *uniquement* pour des raisons de compatibilité avec des normes de base.

Le standard Unicode ne tente pas de coder des caractéristiques de texte comme la langue, la police, la force de corps, l'emplacement, l'œil des caractères (glyphes), etc. Par exemple, il ne conserve pas d'information sur la langue lors du codage de caractères : l'*i grec* français, l'*ypsilon* allemand et le *wye* anglais sont tous représentés par le même code de caractère, U+0057 « Y », il en va de même pour le *zi* (*tseu*) chinois, le *ji* japonais et le *ja* coréen représentés par le même code de caractère, U+5B57 字 .

L'unification des variantes idéographiques suit les principes édictés à la section *11.1, Han*. Quand, selon ces principes, deux formes ne se distinguent que par des différences anodines (*wazoukana*), on ne leur attribue qu'un seul point de code. Dans les autres cas, on leur affecte deux numéros de code distincts.

Caractères de compatibilité. On appelle caractères de compatibilité des caractères qui n'auraient pas été codés si ce n'est à des fins de compatibilité, car ils ne sont, dans une certaine mesure, que des variantes de caractères déjà codés. Les exemples typiques sont les variantes de glyphes que l'on retrouve dans la zone de compatibilité : caractères de demi-chasse, glyphes contextuels arabes, ligatures arabes et ainsi de suite.

La zone de compatibilité contient un grand nombre de caractères de compatibilité, mais l'on trouve de nombreux caractères de compatibilité qui ne sont pas repris dans la zone de ce nom. Parmi les exemples que l'on peut citer : les chiffres romains, comme le « caractère » IV. Il est important de pouvoir identifier les caractères de compatibilité afin que les systèmes architecturés autour d'Unicode puissent les traiter d'une manière uniforme.

Dire d'un caractère qu'il est une variante de compatibilité d'un autre caractère signifie qu'on peut généralement faire correspondre les deux caractères sans autre perte d'information que celle liée au formatage. Cependant, il n'est pas toujours possible d'effectuer de telles correspondances car un grand nombre de caractères de compatibilité n'existent que dans le but de définir une bijection entre Unicode et des jeux de caractères existants. Or une fusion entraînerait une perte d'information considérée comme importante dans le jeu de caractère originel. La liste des transformations de compatibilité est expliquée à la section *15.1, Liste des noms de caractère.* Étant donné que le remplacement d'un caractère par un caractère ou une suite de caractères équivalents et compatibles peut modifier l'information stockée dans un texte, cette opération doit être effectuée avec précaution. Il vaut mieux utiliser ces

correspondances pour spécifier une équivalence valable dans le cadre de tris ou de repérages, et non pour transcoder des textes.

Composition dynamique

Le standard Unicode permet de composer dynamiquement les formes accentuées des lettres et des syllabes hangûl. Combiner des caractères pour créer des formes composées est un processus productif car il est ouvert. On peut donc créer de nouvelles formes à signes modificatifs à l'aide d'une combinaison de caractères de base[1] suivis de signes combinatoires. Ainsi, le tréma « ¨ » peut-il être se joindre à d'autres voyelles ou à un certain nombre de consonnes dans des langues qui utilisent l'écriture latine mais également dans d'autres écritures.

Séquence équivalente

Certains éléments textuels peuvent être codés à l'aide d'une forme précomposée ou d'une composition dynamique. Pour des raisons de compatibilité avec les normes actuelles, on a inclus dans le standard de nombreuses formes précomposées ordinaires comme U+00DC Ü LETTRE MAJUSCULE LATINE U TRÉMA. Le standard fournit pour les formes statiques précomposées l'équivalent canonique qui correspond à la suite composée dynamiquement de ces caractères (cf. également la section *3.6, Décomposition*).

Dans bien des cas, on considère équivalentes des suites différentes de caractères Unicode. Par exemple, un caractère précomposé peut être représenté à l'aide d'une suite de caractères (cf. *Figures 2-5 et 2-10*).

Figure 2-5. Séquences équivalentes

B + Ä → B + A + ◌̈

LJ + A → L + J + A

Le standard Unicode ne se prononce pas en faveur d'une suite particulière dans ces cas, toutes les suites illustrées sont équivalentes. Certains systèmes peuvent décider de normaliser le texte Unicode en préférant une certaine suite, en normalisant, par exemple, les suites de caractères composés en des caractères précomposés ou vice-versa. Il n'est donc pas sûr qu'une distinction faite entre deux séquences équivalentes non identiques par des applications ou des utilisateurs soit portable. (Pour des conseils de mise en œuvre, cf. la section *5.7, Normalisation* et le chapitre *7, Formes de normalisation Unicode.*)

Convertibilité

L'identité d'un caractère est conservée afin de garantir l'échange de données textuelles avec différentes normes de base : normes nationales, internationales ou standards de l'industrie. Lorsque des variantes de forme (ou même la même forme) se sont vues affectées des numéros de caractère différents dans une norme de base, Unicode les garde séparées. Cette décision garantit l'existence d'une correspondance bijective entre le standard Unicode et les normes de base.

[1] Par exemple, dans le cas des diacritiques doubles qui recouvrent les deux caractères de base qui précèdent.

Le standard Unicode garantit une convertibilité exacte entre son répertoire et un ensemble de normes bien établies en mai 1993. En général, toute valeur de code dans une autre norme correspond à une valeur de code en Unicode. Cependant, parfois, une valeur de code dans une autre norme correspondra à une suite de valeur de code Unicode ou vice-versa. Les conversions entre du texte Unicode et du texte codé dans un autre jeu se font habituellement à l'aide de tableaux de correspondance explicites. (Cf. également la section *5.1, Transcodage vers d'autres normes et standards*.)

2.3 Modèle de caractères

Le modèle de caractère défini dans le rapport technique d'Unicode n° 17 comprend cinq niveaux que l'on peut résumer de la sorte :

1. répertoire de caractères abstraits
   - ensemble des caractères à coder, c'est-à-dire un alphabet ou un jeu de symboles
2. jeu de caractères codés
   - fonction qui attribue un numéro aux caractères abstraits du répertoire
3. forme de stockage de caractères
   - relation entre un ensemble de numéros de caractères utilisés dans un jeu de caractères codés et un ensemble de suites d'unités de stockage
4. mécanisme de sérialisation de caractères
   - correspondance entre des unités de stockage et des suites d'octets sérialisés.
5. surcodage de transfert
   - transformation réversible des données codées, celles-ci peuvent ne pas contenir de données textuelles.

La figure 2-6 illustre le rapport qui existe en les quatre premiers niveaux du modèle :

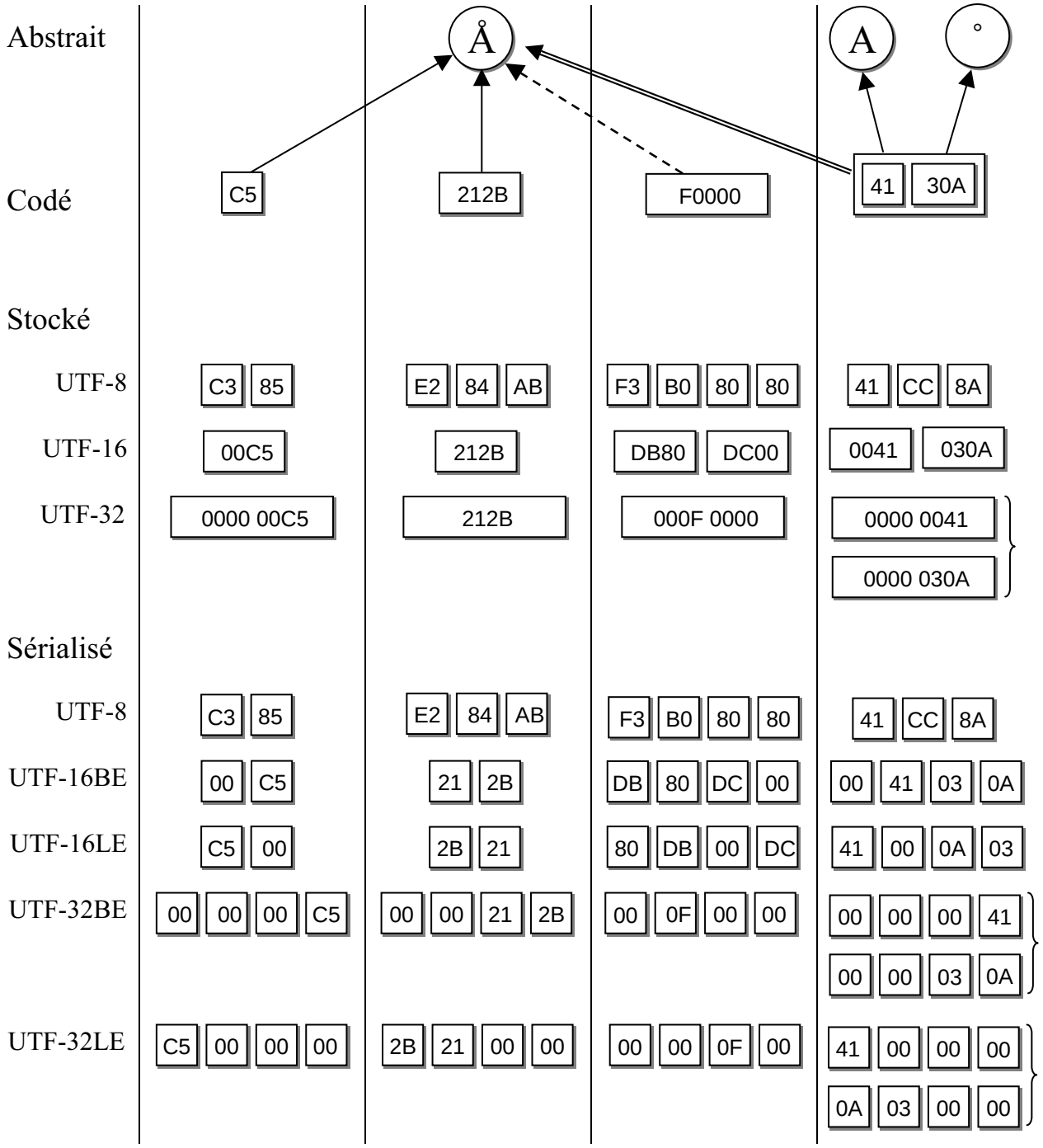
- Les flèches pleines relient les caractères codés (également appelés caractères numérotés) aux caractères abstraits qu'ils représentent et codent.

La flèche en pointillé représente une variante hypothétique d'usage privé du caractère A ROND EN CHEF (la valeur $F0000_{16}$ est située dans la zone d'usage privé).

- La flèche creuse indique une suite de caractères codés qui représente sous forme décomposée le caractère abstrait, aucun des deux caractères codés ne le code donc directement.

- Les rangées *stocké* et *sérialisé* représentent différentes formes stockées et sérialisées de caractères codés.

Figure 2-6. Mécanismes de codage de caractères



Répertoire de caractères abstraits

Un répertoire est un ensemble de caractères abstraits destinés à être codés, il s'agit habituellement d'un alphabet connu ou d'un ensemble de symboles. Le mot abstrait signifie simplement que ces objets sont définis par convention, comme le sont les 26 lettres de l'alphabet latin, les majuscules ou les minuscules. Il est bon de rappeler que le répertoire comprend des caractères, et non des glyphes.

Un répertoire est un ensemble non ordonné dont il existe deux types : *fermé* ou *ouvert*. Pour la plupart des codages de caractères existants, le répertoire est fermé (et souvent petit). Une fois le répertoire établi, celui-ci ne change plus. On considère habituellement que l'ajout d'un nouveau caractère abstrait à un répertoire fermé entraîne la définition d'un nouveau répertoire.

Unicode, par contre, est un répertoire essentiellement ouvert. Étant donné qu'Unicode se veut un codage universel, tout caractère abstrait qui pourra jamais être codé est membre potentiel du jeu à coder, que l'on connaisse actuellement ce caractère ou non.

Jeu de caractères codés

On définit un jeu de caractères codés comme une correspondance entre un ensemble de caractères abstraits et un ensemble d'entiers non négatifs. Ce dernier ensemble peut ne pas être contigu. On dit qu'un caractère abstrait est défini dans un jeu de caractères donné si un numéro de caractère existe pour ce caractère. Ce caractère abstrait est alors appelé un *caractère codé*. Le concept de valeur scalaire Unicode (cf. D27 au chapitre *3, Conformité*) correspond explicitement à ce numéro de caractère.

On désigne parfois les jeux de caractères codés sous les noms de codage de caractères, de répertoire codé de caractères, de définition de jeu de caractère ou de page de code.

Exemples de jeu de caractères codés :

- JIS X 0208
  - attribue une paire d'entiers, connus sous le nom de point *kouten*
- ISO/CEI 8859-1
- ISO/CEI 8859-2
  - répertoire différent de 8859-1
- page de code 037
  - même répertoire que 8859-1, numéros différents
- le standard Unicode 2.0
- ISO/CEI 10646 : 1993 + amendements 1 à 7
  - exactement le même répertoire et correspondance caractère-numéro qu'Unicode 2.0
- le standard Unicode 3.1
- ISO/CEI 10646 : 2000
  - répertoire et correspondance caractère-numéro identique à Unicode 3.0

L'intervalle des entiers non négatifs qui peuvent être affectés aux caractères abstraits définit un concept connexe d'*espace de code*. Les limites des différents espaces de code dépendent fortement de la forme de stockage des caractères (voir ci-dessous) puisque la correspondance entre les caractères abstraits et leurs numéros n'est pas arbitraire, mais s'effectue en fonction

d'une forme particulière de stockage. Exemples d'espace de code : 0..7F, 0..FF, 0..FFFF, 0..10FFFF,0..7FFFFFFF, 0..FFFFFFFF. Les espaces de code peuvent avoir une structure assez élaborée, certains peuvent être continus alors que d'autres interdisent plusieurs intervalles particuliers de valeurs.

Forme stockée de caractères

Une forme stockée de caractères définit la relation entre un ensemble de numéros de caractères utilisés dans un jeu de caractères codés et un ensemble de suites d'unités de stockage. Une unité de stockage est un entier d'une certaine largeur (exemples : octet ou seizet) qui sert d'unité de base à l'expression des numéros de caractère dans la mémoire d'un ordinateur. Cette forme stockée définit la représentation réelle des caractères codés comme données informatiques. Les suites d'unités de stockage ne doivent pas nécessairement avoir toutes la même longueur.

- Une forme stockée de caractères dont les suites sont toutes de même longueur est dite de *largeur fixe*.
- Une forme stockée de caractères dont les suites n'ont pas toutes la même longueur est dite de *largeur variable*.

On dit qu'une forme stockée de caractères est définie pour un jeu particulier de caractères codés lorsqu'elle fait correspondre un caractère stocké à tout caractère de ce jeu. Souvent, on n'a défini qu'une seule forme stockée pour un jeu de caractères codés. Dans ces cas-là, on ne spécifie que la forme stockée de caractères. Le jeu de caractères codés est donc implicite et défini par un rapport implicite entre les suites stockées de caractères et les numéros.

Il existe une grande variété de formes stockées. Certaines sont propres à Unicode et à l'ISO/CEI 10646, d'autres constituent des modèles réutilisés par des centaines de jeux de caractères.

Voici quelques exemples de formes stockées à largeur fixe :

- 7 bits
  - chaque numéro de caractère est représenté par un septet (quantité à 7 bits). Exemple : ISO 646.
- 8 bits G0/G1
  - chaque numéro de caractère est représenté par un octet, avec certaines contraintes quant à l'utilisation des zones C0 et C1[2].
- 8 bits
  - chaque numéro de caractère est représenté par un octet sans contrainte sur l'utilisation de la zone C1.
- 8 bits EBCDIC
  - chaque numéro de caractère est représenté par un octet avec les conventions de l'EBCDIC plutôt que celles de l'ASCII.
- 16 bits (UCS-2)
  - chaque numéro de caractère est représenté par un seizet.
- 32 bits (UCS-4, UTF-32)

[2] Les deux zones de caractères de commande de l'ISO 6429 :1992, U+0000..U+001F pour C0, U+0080..U+009F pour C1.

  - chaque numéro de caractère est représenté par une quantité sur 32 bits. En UTF-32, l'espace de code est arbitrairement limité à 0..10FFFF.

Exemples de formes stockées de largeur variable :

- UTF-8
  - utilisé uniquement avec Unicode et l'ISO/CEI 10646 : un mélange d'une à quatre unités de stockage (des octets ici) pour Unicode et théoriquement d'une à six unités de stockage pour l'ISO/CEI 10646.
- UTF-16
  - utilisé uniquement avec Unicode et l'ISO/CEI 10646 : un mélange d'une à deux unités de stockage (seizets dans ce cas-ci).

La forme stockée de caractères constitue une des relations fondamentales auxquelles les logiciels internationalisés s'intéressent : combien d'unités de stockage utilise-t-on pour chaque caractère ? On exprime cette relation en termes de nombre d'octets par caractère. Avec l'introduction d'UCS-2, d'UTF-16, d'UCS-4 et d'UTF-32 et de leurs unités de stockage plus larges qu'un octet pour Unicode et l'ISO/CEI 10646, cette relation s'exprime à l'aide de deux paramètres : la largeur des unités de stockage et le nombre d'unités de stockage utilisé pour représenter chaque caractère.

UTF-16

Cette forme de stockage du standard Unicode est à 16 bits : chaque caractère se voit assigner un seizet unique, hormis les caractères codés dans les plans supérieurs du JUC et qui sont eux codés à l'aide de deux seizets d'indirection. La forme de stockage à 16 bits d'Unicode est identique au format transformé UTF-16 de l'ISO/CEI 10646. En UTF-16, les numéros de caractères inférieurs à 65 536 sont stockés sous la forme d'un seul seizet, ceux supérieurs à 65 535 sont stockés à l'aide de deux seizets (dits *d'indirection*). Pour plus de renseignements sur les seizets d'indirection, se reporter à la section *3.7, Seizets d'indirection.*

Grâce aux seizets d'indirection, UTF-16 permet de coder plus d'un million de caractères graphiques dans une forme compatible avec les caractères sur 16 bits. Ce mécanisme permet de représenter un très grand nombre de caractères sur des plates-formes qui définissent les caractères en fonction de seizets. Les seizets d'indirection ont été conçus comme une extension simple et efficace fonctionnant tout aussi bien avec les mises en œuvre préexistantes. Cette extension permet également d'éviter les écueils des codages multioctets. Cf. la section *5.4, Traitement des paires d'indirection* pour plus de renseignements.

UTF-8

Afin de satisfaire les besoins des systèmes architecturés autour de l'ASCII ou d'autres jeux de caractères à un octet, le standard Unicode définit une forme de stockage supplémentaire : l'UTF-8. Il s'agit d'une forme de stockage à largeur variable transparente pour les systèmes ASCII. L'utilisation d'UTF-8 est pleinement conforme au standard Unicode et à la norme ISO/CEI 10646.

Les logiciels et les algorithmes existants supposent souvent que les données textuelles sont représentées par des suites d'octets. Avant d'utiliser des données textuelles avec de tels systèmes, il faut transformer les valeurs de caractères Unicode et les paires de seizets d'indirection en une suite d'un ou plusieurs octets qui représentent la même information, mais dont les valeurs sont restreintes.

UTF-8 est un codage constitué de suites d'octets de longueur variable ; les bits de poids le plus fort d'un octet indiquent la position de celui-ci dans la suite d'octets. Le *tableau 3-1* de la section *3.8, Transformation* montre comment les bits d'un numéro de caractère Unicode sont répartis parmi les octets UTF-8. Toute suite d'octets qui ne suit pas ce modèle est une suite mal formée d'octets. La forme de stockage UTF-8 assure la transparence de toutes les valeurs de code ASCII (0..127). En outre, ces mêmes valeurs n'apparaissent pas dans les octets transformés sauf en tant que représentation directe de ces mêmes valeurs ASCII. Les caractères de l'intervalle ASCII (U+0000..U+007F) sont stockés à l'aide d'un seul octet, la plupart des caractères issus des alphabets ou syllabaires sont stockés à l'aide de deux octets, les autres valeurs inférieures à $FFFF_{16}$ avec trois octets et, enfin, les valeurs de 10000_{16} à $10FFFF_{16}$ à l'aide de quatre octets.

Autres caractéristiques importantes de l'UTF-8 :

- Conversion efficace à partir de ou vers un texte codé en UTF-16.
- Le premier octet indique le nombre d'octets qui suit dans le cas de séquence multioctets, ceci permet une analyse rapide du texte vers l'avant.
- Recherche rapide du début de tout caractère, peu importe où l'on commence la recherche dans un flux de données. Il suffit de consulter au plus quatre octets en amont pour reconnaître aisément le premier octet.
- UTF-8 est un mécanisme de stockage relativement compact en termes d'octets
- Un tri binaire des chaînes UTF-8 donne le même résultat qu'un tri binaire effectué sur les valeurs scalaires Unicode correspondantes.
- Dans le cas où l'on utiliserait des seizets d'indirection, un tri binaire de ces chaînes UTF-8 donne le même résultat qu'un tri binaire effectué sur les valeurs UTF-16 correspondantes.

On trouvera des exemples de code-source de transformation de données UTF-16 en UTF-8 sur le disque qui accompagne cet ouvrage.

UTF-32

Dans certains cas, on pourra préférer l'utilisation d'une forme de stockage sur 32 bits, où chaque numéro de caractère Unicode (également appelé valeur scalaire Unicode) correspond à une unité codée sur 32 bits. Même les applications qui n'utilisent pas cette forme voudront sans doute convertir des données stockées à partir de et vers ce format à des fins d'interopérabilité.

Quelques caractéristiques importantes de cette forme de stockage :

- UTF-32 est restreint aux valeurs appartenant à l'intervalle $0..10FFFF_{16}$ qui correspond parfaitement à l'étendue des caractères définie par le standard Unicode (et utilisée par d'autres standards tels qu'XML).
- Le standard Unicode ajoute un certain nombre de contraintes de conformité, en sus de celles précisées par l'ISO/CEI 10646, plus particulièrement quant à la sémantique des caractères (cf. le chapitre *3, Conformité*). Déclarer un texte UTF-32 plutôt qu'UCS-4 permet de spécifier l'application des règles sémantiques propres à Unicode.
- De par sa forme de stockage à largeur fixe (sur 32 bits), la valeur numérique d'un caractère Unicode en UTF-32 correspond toujours à sa valeur scalaire Unicode.

Mécanisme de sérialisation de caractères

Un mécanisme de sérialisation définit une correspondance entre des unités de stockage et des suites d'octets sérialisés. Cette transformation permet de définir des formes d'échange de données persistantes intermachine dont les unités de stockage sont plus grandes que l'octet. Il se peut alors qu'une permutation des octets s'avère nécessaire afin d'ordonner les données selon la polarité canonique des octets pour la plate-forme en question.

En particulier :

- La plupart des formes de stockage à largeur fixe sur un septet ou octet ont une projection triviale vers un mécanisme de sérialisation de caractères : chaque septet ou octet se transforme en un octet de même valeur.
- La plupart des formes de stockage à largeur variable ne font que sérialiser les suites d'unités de stockage en octets.
  - UTF-8, dont les unités de stockage sont constituées d'un octet, suit ce principe.
  - UTF-16, dont les unités de base sont des seizets, doit préciser l'ordre des octets à adopter pour la sérialisation. C'est là la différence entre UTF-16BE, dont les deux octets de seizet sont sérialisés en ordre gros-boutien, et l'UTF-16LE qui les sérialisent en ordre petit-boutien.
  - UTF-32, dont les unités de base sont des quantités à 32 bits, doit préciser l'ordre des octets à adopter lors de la sérialisation. Tout comme UTF-16, il existe donc deux mécanismes de sérialisation : UTF32-BE sérialise les octets en ordre gros-boutien alors qu'UTF-32LE les sérialise en ordre petit-boutien. UTF32-BE est structurellement identique à l'UCS-4 tel que défini par l'ISO/CEI 10646-1 :2000.

Il est important de ne pas confondre forme stockée de caractères et mécanisme de sérialisation de caractères :

1. La forme stockée de caractères fait correspondre des unités de stockage aux numéros de caractère, alors que le mécanisme de sérialisation de caractères fait correspondre ces unités de stockage à des octets sérialisés.
2. Le mécanisme de sérialisation doit préciser de quelle façon les unités de stockage sont transformées en suites d'octets.

Exemples de mécanisme de sérialisation :

- Unicode 3.1 a cinq mécanismes de sérialisation de caractères : UTF-8, UTF-16BE, UTF-16LE, UTF-32BE et UTF-32LE
- Unicode 1.1 possédait également trois mécanismes de sérialisation de caractères : UTF-8, UCS-2BE et UCS-2LE.
- Les jeux de caractères de la famille ISO 2022 (ISO-2022-JP, ISO-2022-KR, etc.) qui incorporent au flux de caractères des séquences d'échappement.

Surcodage de transfert

Un surcodage de transfert est une transformation réversible de données codées qui peut ou non comprendre des données textuelles représentées à l'aide d'un ou de plusieurs mécanismes de sérialisation de caractères.

Généralement, on met au point des surcodages de transfert pour plusieurs raisons.

- Éviter l'utilisation de certaines valeurs d'octets qui pourraient déboussoler certains protocoles de transmission ou de stockage, c'est la fonction de base64, uuencode, BinHex et du *quoted-printable* de MIME.

- Appliquer de manière formelle différents algorithmes de compression aux données afin de réduire le nombre de bits transmis, c'est la fonction de pkzip, gzip, winzip, etc.

SCSU

SCSU (cf. rapport technique d'Unicode n° 6 : *A standard Compression Scheme for Unicode*) peut également être conçu comme un surcodage de transfert.

Voici quelques exemples de données surcodées à l'aide de SCSU, on reconnaîtra les caractères de la figure 2-6 :

Codé	Surcodé pour transfert (SCSU)	Commentaires
C5	C5	l'état initial de SCSU est Latin-1
212B	OE 21 2B	OE signifie *citer Unicode*
F0000	OE DB 80 OE DC 00	citer Unicode + 2 seizets d'indirection
41 30A	41 03 0A	03 signifie *passage à Win 3 non verrouillé*

2.4 Affectation

Toutes les valeurs de code du standard Unicode sont d'une égale facilité d'accès ; l'affectation précise des caractères est de peu de conséquence du point de vue informatique. Cependant, afin de rendre plus commode leur consultation, les codes ont été regroupés par catégorie linguistique ou fonctionnelle. Ces groupements procurent des avantages supplémentaires dans la mise en œuvre de plusieurs techniques de compression.

Zones d'affectation du PMB

Les figures 2-7a et 2-7b donnent un aperçu de l'allocation de l'espace de code Unicode. Pour plus de commodité, l'espace de code du standard Unicode se divise en plusieurs zones, elles-mêmes subdivisées en blocs de caractères :

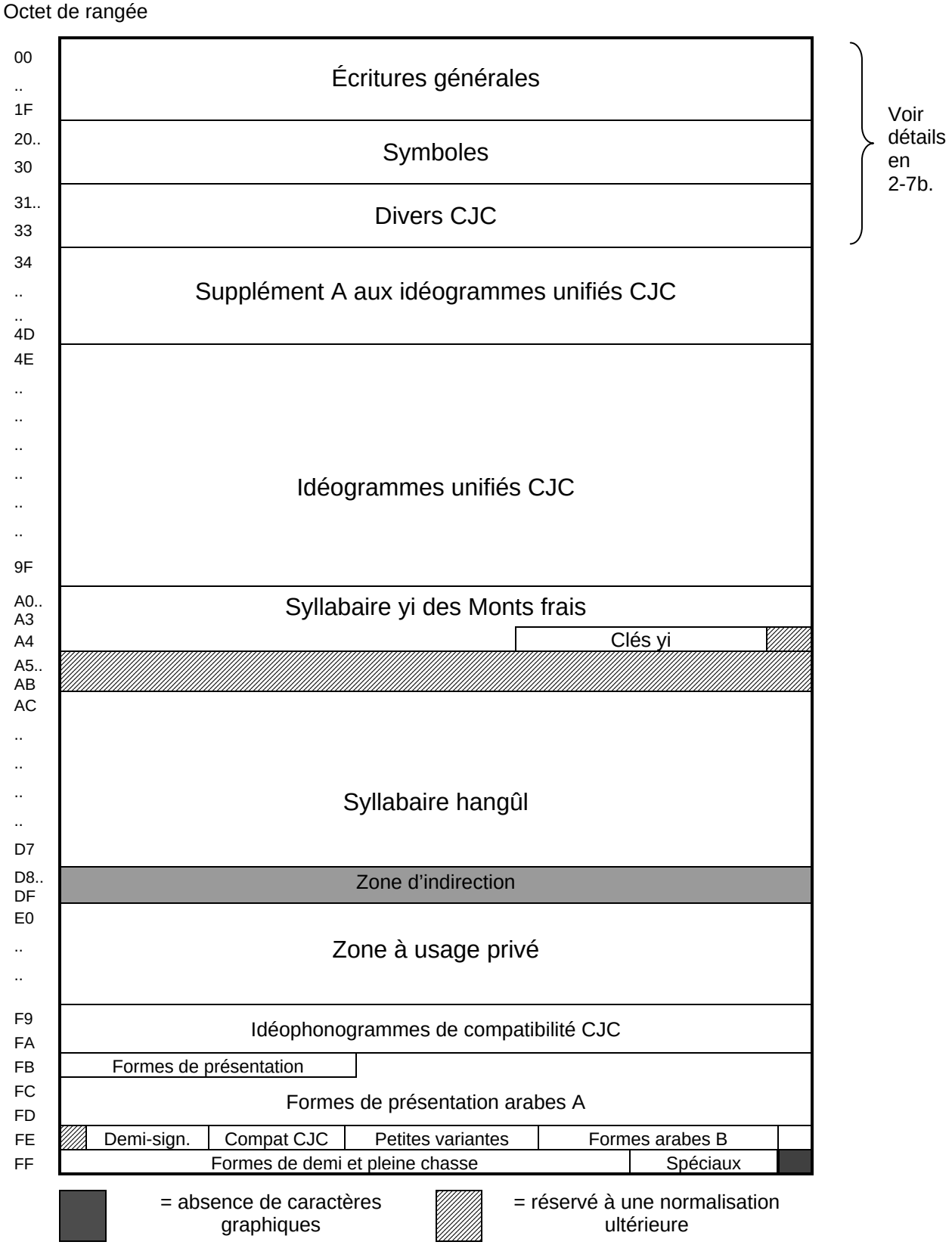
- La *zone des écritures générales* qui comprend les écritures alphabétiques et syllabiques codées à l'aide de petits jeux de caractères, comme le latin, le cyrillique, le grec, l'hébreu, l'arabe, la dévanâgarî ou le thaï.
- La *zone des symboles* qui inclut une panoplie de symboles mathématiques, chimiques, techniques, de ponctuation et d'autres spécialités.
- La *zone des symboles et des signes phonétiques CJC* qui rassemble des signes de ponctuation, des symboles, des clés (encore appelés radicaux) et des signes phonétiques utilisés pour le chinois, le japonais et le coréen.

- La *zone des idéogrammes unifiés CJC* consiste en 27 484 idéophonogrammes CJC unifiés.
- La *zone du syllabaire yi* des Monts frais comprend 1 165 syllabes et 50 radicaux yi.
- La *zone du syllabaire hangûl* regroupe 11 172 syllabes coréennes hangûl précomposées.
- La *zone d'indirection* divisée en 1 024 seizets d'indirection inférieurs et 1 024 seizets d'indirection supérieurs qui seront utilisés dans des versions ultérieures d'Unicode pour accéder à un potentiel de plus d'un million de codes supplémentaires.
- La *zone à usage privé* compte 6 400 positions de code mis à la disposition des utilisateurs ou des éditeurs de logiciels pour qu'ils puissent définir leurs propres caractères.
- La *zone de compatibilité* et de signes spéciaux comprend de nombreux caractères provenant de jeux de caractères bien établis de l'industrie informatique ou de normes nationales qui ont une autre représentation dans le codage Unicode proprement dit (à savoir les autres zones), cette zone contient également de nombreux caractères spéciaux.

L'affectation des caractères à une zone particulière reflète l'évolution du standard Unicode et ne vise pas à définir une utilisation particulière des caractères dans les mises en œuvre. Ainsi, de nombreux caractères ne font partie du standard qu'à des fins de compatibilité avec d'autres standards ou normes, mais ils ne sont pas codés dans la zone de compatibilité ; de nombreux symboles et signes de ponctuation d'utilité générale se retrouvent dans la zone des symboles et des signes phonétiques CJC alors que les *jamos jointifs* hangûl se retrouvent dans la zone des écritures générales.

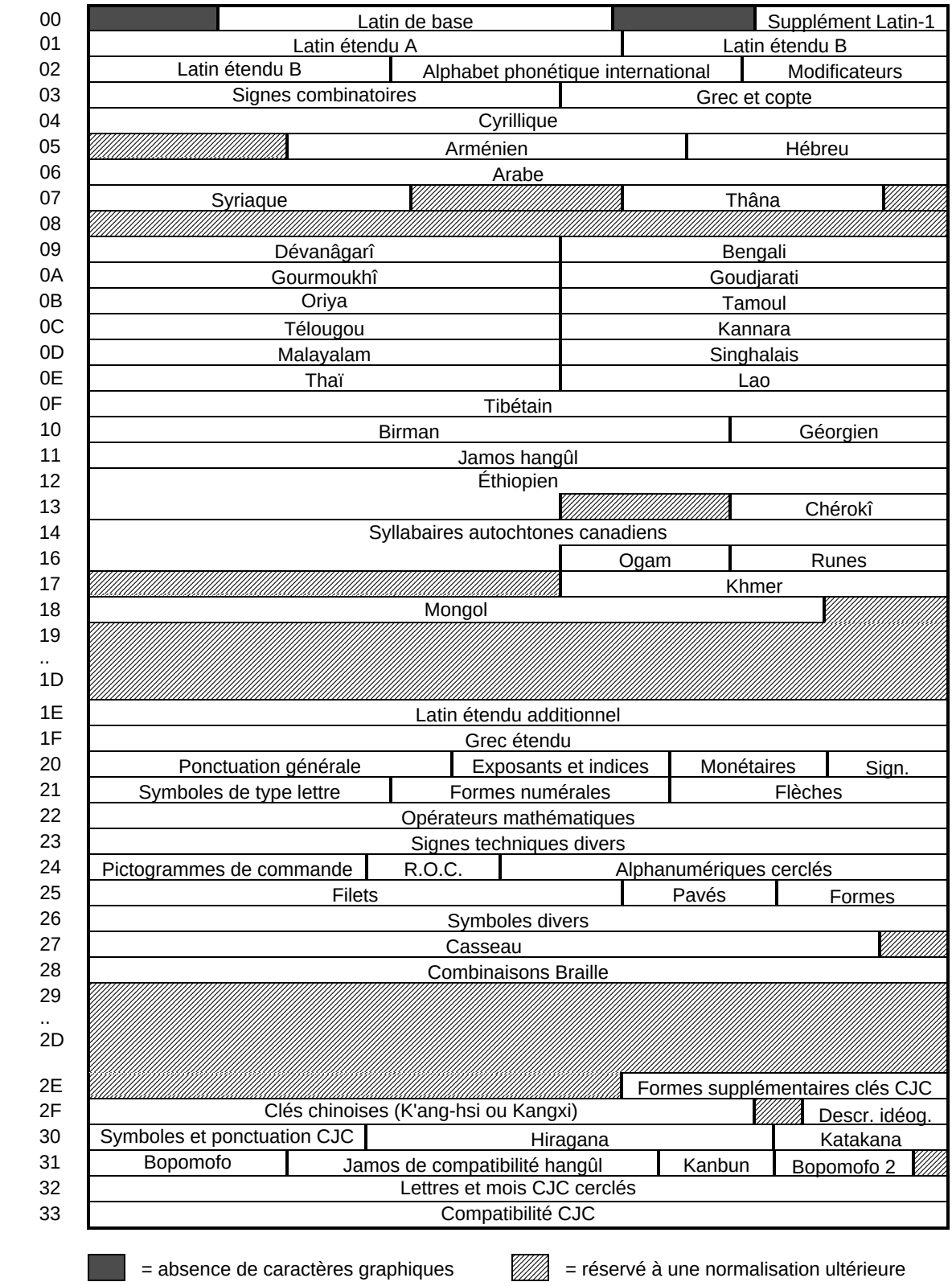
La zone à usage privé donne au standard Unicode une marge de manœuvre suffisante et correspond à une pratique courante dans les normes actuelles. Pour qu'un échange de données codées dans la zone à usage privé puisse avoir du sens, l'émetteur et le récepteur doivent s'être mis d'accord au préalable sur l'interprétation des codes puisque Unicode n'en attribue aucune.

Figure 2-7a. Vue d'ensemble des zones d'affectation du PMB



NOTE : Les séparations verticales à l'intérieur d'une rangée sont approximatives.

Figure 2-7b. Vue d'ensemble des rangées 00 à 33 du PMB



NOTE : Les séparations verticales à l'intérieur d'une rangée sont approximatives.

Attribution des caractères graphiques au sein de l'espace de code

Les traits prédominants de l'attribution de caractères au sein de l'espace de code Unicode peuvent se résumer ainsi :

- Lorsqu'il existe une seule norme *établie* pour une écriture donnée, le standard Unicode suit généralement l'ordre relatif défini pour les caractères de cette écriture.
- Les 256 *premiers* codes correspondent exactement à la disposition de l'ISO/CEI 8859-1 (Latin 1) dont les 128 premiers codes correspondent à l'ASCII à 7 bits (ISO/CEI 646 IRV).
- Les caractères aux caractéristiques communes sont regroupés en ensembles contigus. Ainsi, le bloc principal de caractères arabes s'inspire-t-il de l'ISO/CEI 8859-6. Les caractères de l'écriture arabe utilisés en perse, en ourdou et en d'autres langues, mais absents de l'ISO/CEI 8859-6, ont été attribués à la suite du bloc principal des caractères arabes. Les écritures s'écrivant de droite à gauche sont regroupées.
- Dans la mesure du possible, les écritures ne dépassent pas la limite des 128 ou des 1 024 positions de code.
- Les codes qui représentent des lettres, de la ponctuation, des symboles et des diacritiques utilisés par plusieurs langues ou écritures sont regroupés à plusieurs endroits.
- Le standard Unicode n'a pas la prétention de faire correspondre l'attribution des codes de caractère à l'ordre lexicographique ni à la casse d'une langue.
- Les idéophonogrammes CJC unifiés sont divisés en deux sections, chacune est disposée selon l'ordre idéographique han défini à la section *11.1, Han*. Celui-ci correspond, *grosso modo*, à l'ordre radical-nombre de traits.

Caractères non graphiques, codes réservés et non affectés

Tous les numéros sauf ceux mentionnés ci-dessous sont réservés aux caractères graphiques. Les numéros non affectés dans cette version du standard Unicode pourront être attribués, lors d'une normalisation ultérieure, à des caractères de n'importe quelle écriture.

- Soixante-cinq numéros (U+0000..U+001F et U+007F..U+009F) sont expressément réservés à des codes de commande. Parmi ces codes de commandes, *nul* (U+0000) peut servir à indiquer la fin d'une chaîne dans le langage C, *tabulation horizontale* (U+0009) conserve sa signification habituelle et les autres sont interprétés à la lumière de l'ISO/CEI 6429 (cf. les sections *2.8, Commandes et suites de commandes* et *14.1, Codes de commande*).

- On ne codera jamais de caractères à l'aide des deux numéros suivants : U+FFFF et U+FFFE. Le numéro U+FFFF peut servir de sentinelle interne et ne doit jamais être transmis ou stocké dans du texte brut. Le numéro U+FFFE, quant à lui, est également réservé, sa présence indique des données Unicode aux octets inversés (cf. la section *14.6, Codes spéciaux*).

- Il existe une zone contigüe réservée à l'usage privé. Le standard Unicode ne définira jamais de caractères dans cette zone. Ces numéros peuvent donc librement être

attribués à des caractères quel qu'en soit l'usage. Mais pour qu'un échange de données codées dans cette zone réussisse, l'émetteur et le récepteur doivent se mettre d'accord sur leur interprétation.

- En outre, 2 048 numéros sont réservés comme seizets d'indirection, ils servent au mécanisme d'extension (cf. la section 3.7, *Seizets d'indirection*).

2.5 Direction d'écriture

Chaque système d'écriture peut obéir à des conventions différentes quant à l'ordre dans lequel il faut disposer les caractères sur une ligne, une page ou un écran. On désigne ces conventions sous le nom de *directionalité* de l'écriture. Ainsi, dans le système d'écriture latin, les caractères sont-ils disposés horizontalement de gauche à droite pour former des lignes qui elles-mêmes sont disposées de haut en bas.

Dans les écritures sémitiques, comme l'hébreu et l'arabe, les caractères s'écrivent de droite à gauche, bien que leurs chiffres s'écrivent dans l'autre sens, ce qui en fait des écritures intrinsèquement bidirectionnelles. Les écritures de gauche à droite et de droite à gauche s'écrivent souvent ensemble. La formation de leurs lignes soulève plus de difficultés. Le standard Unicode définit un algorithme qui précise la disposition d'une ligne. Cf. la section *3.12, Comportement bidirectionnel* pour un supplément d'information.

Les écritures d'Extrême-Orient s'écrivent souvent de haut en bas en lignes verticales. Les lignes sont disposées de droite à gauche, sauf pour le mongol. La plupart des caractères ont la même apparence et orientation qu'ils soient affichés horizontalement ou verticalement, cependant beaucoup de signes de ponctuation changent d'apparence quand on les affiche verticalement. Dans un contexte vertical, les lettres et mots d'autres écritures pivotent de 90 degrés pour qu'on puisse aussi les lire de bas en haut. En d'autres termes, les lettres provenant des écritures de gauche à droite pivotent dans le sens horaire alors que les lettres des écritures de droite à gauche pivotent dans le sens anti-horaire.

À l'inverse du cas bidirectionnel, la décision de disposer un texte verticalement ou horizontalement est considérée comme une variante de style. C'est pourquoi Unicode ne prévoit pas de commande de directionalité pour indiquer ce choix.

On trouve parmi les écritures historiques d'autres directionalités. Certains textes antiques numides, par exemple, sont écrits de bas en haut. Les textes hiéroglyphiques peuvent adopter plusieurs directions différentes au sein d'une même ligne.

Les plus anciens textes grecs utilisaient un système appelé *boustrophédon* (sillon de bœuf). En boustrophédon, les caractères forment des lignes horizontales qui s'écrivent alternativement de gauche à droite et de droite à gauche en imitant le va-et-vient du bœuf qui laboure un champ. Selon la direction de chaque ligne, on inverse les lettres par symétrie verticale.

Le boustrophédon n'intéresse à peu près que les érudits qui désirent reproduire fidèlement le contenu visuel de textes anciens. Le standard Unicode ne prévoit pas de moyen de représenter le boustrophédon. On peut cependant écrire des textes invariables en boustrophédon en forçant la directionalité de chaque ligne et les passages à la ligne.

2.6 Caractères combinatoires

Caractères combinatoires. Les caractères destinés à s'afficher par rapport à un caractère de base sont représentés dans les tableaux de codes de caractère au-dessus, en dessous ou au travers d'un cercle en pointillé. Ce cercle pointillé représente le caractère de base en question. Les annotations de ces caractères qui suivent les tableaux de codes ou les listes de propriétés de caractère (au chapitre 15) comportent également la mention « combinatoire », « sans chasse » ou « à chasse nulle ». Lors du rendu, les glyphes qui représentent ces caractères doivent se placer par rapport au glyphe qui représente le caractère de base dans un certain ordre. Le standard Unicode distingue deux types de signes combinatoires : avec ou sans chasse. Les caractères combinatoires sans chasse (encore appelés à chasse nulle) n'occupent pas de place par eux-mêmes. À l'affichage, la combinaison du caractère de base et du caractère à chasse nulle peut occuper plus d'espace latéral que le caractère de base seul. Ainsi, un « î » chasse légèrement plus qu'un simple « i ». Les propriétés de chasse ou d'absence de chasse d'un caractère combinatoire sont spécifiées dans la *Base de données de caractères Unicode.*

Diacritiques. Les diacritiques constituent la classe principale des caractères combinatoires sans chasse utilisée avec les alphabets européens. Dans le standard Unicode, la définition du terme « diacritique » est relativement vague et comprend à la fois les accents ainsi que d'autres signes sans chasse.

Tout caractère de base peut se voir adjoindre n'importe quel diacritique quelle que soit l'écriture de ce caractère. Il existe un bloc réservé aux diacritiques pour symboles destinés habituellement à des caractères de base de type symbole. On retrouve dans les différents blocs du standard des caractères combinatoires utilisés principalement avec les écritures représentées par ces blocs. Comme pour les autres caractères, l'affectation d'un caractère combinatoire à un bloc ne signale que sa principale utilisation, elle ne définit ni ne limite l'ensemble des caractères auxquels ce caractère combinatoire s'applique. *Dans le standard Unicode, toutes les suites de caractères sont permises.*

Autres caractères combinatoires. Certaines écritures, comme l'arabe ou l'hébreu ainsi que les écritures de l'Inde ou du Sud-Est asiatique, possèdent des caractères combinatoires avec ou sans chasse. Beaucoup de ces signes combinatoires codent des lettres-voyelles.

Suite de caractères de base et diacritiques

Dans le standard Unicode, tous les caractères combinatoires doivent suivre le caractère de base qu'ils qualifient. La suite de caractères Unicode U+0061 a LETTRE MINUSCULE LATINE A + 0308 ◌̈ DIACRITIQUE TRÉMA + 0075 u LETTRE MINUSCULE LATINE U code sans équivoque « äu » et non « aü ».

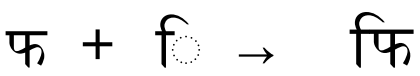
La convention d'ordonnancement utilisée par le standard Unicode correspond à l'ordre logique des diacritiques arabes et indiens dont la grande majorité suit (logiquement ou phonétiquement) les caractères de base par rapport auxquels ils se placent. Cette convention se conforme à la manière dont les techniques de police modernes rendent les formes graphiques (glyphes) sans chasse afin de simplifier le passage de l'ordre en mémoire des caractères à celui dans lequel la police est rendue. Il s'agit d'une convention différente de celle utilisée dans la norme bibliographique ISO 5426.

Une suite constituée d'un caractère de base et d'un ou plusieurs caractères combinatoires possède habituellement les mêmes propriétés que le caractère de base. Ainsi, « A » suivi de « ^ » a les mêmes propriétés que « Â ». Selon le contexte, un diacritique englobant peut

conférer la propriété de symbole au caractère englobé. Ce concept fait l'objet d'une plus ample description aux sections *3.10, Mise en ordre canonique* et *3.12, Comportement bidirectionnel*.

Dans les tableaux consacrés aux écritures de l'Inde, certaines voyelles sont représentées à la gauche d'un cercle pointillé (cf. *figure 2-8*). Il faut distinguer attentivement ce cas spécial de celui des diacritiques habituels. Ces signes-voyelles doivent être rendus à la gauche de la consonne ou du groupe consonantique même si logiquement ils suivent la consonne dans leur représentation Unicode. On dit que ces signes-voyelles sont antéposés. La norme nationale indienne ISCII code également ces voyelles dans l'ordre phonétique, et non dans l'ordre visuel.

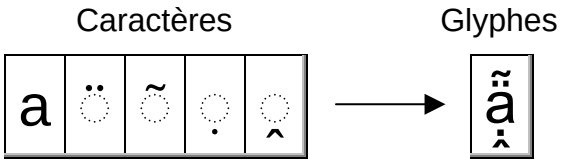
Figure 2-8. Signe vocalique indien antéposé



Caractères combinatoires multiples

Il arrive que plusieurs diacritiques s'adjoignent à un seul caractère (cf. *figure 2-9*). Le standard Unicode ne limite pas le nombre de caractères combinatoires qui peut suivre un caractère de base. On trouvera ci-dessous un bref examen de la question relative au traitement des diacritiques multiples. (Cf. le chapitre *3, Conformité*, pour un algorithme formel.)

Figure 2-9. Empilement de diacritiques



Si les caractères combinatoires peuvent avoir un effet typographique l'un sur l'autre — par exemple un U+0304 ◌̄ DIACRITIQUE MACRON et un 0308 ◌̈ DIACRITIQUE TRÉMA — alors l'ordre des caractères codés détermine l'ordre d'affichage (cf. *figure 2-10*). Les diacritiques sont disposés à tour de rôle à partir du glyphe correspondant au caractère de base pour ensuite s'en éloigner. Les caractères combinatoires placés au-dessus d'un caractère de base s'empilent verticalement, en commençant par le premier dans l'ordre de stockage logique. On poursuit l'empilement vers le haut d'autant de signes qu'il est nécessaire selon le nombre de codes de caractère qui suivent le caractère de base. La situation est inversée pour les caractères combinatoires placés sous un caractère de base : les caractères combinatoires s'empilent vers le bas.

En thaï, il existe des cas où plusieurs caractères combinatoires sont placés au-dessus du caractère de base : les consonnes peuvent être surmontées d'une des voyelles U+0E34 à U+0E37 (◌̂, ◌̄, ◌̌ et ◌̃), elle-même surmontée d'une des quatre marques de ton U+0E48 à U+0E4B (◌̈, ◌̋, ◌̆ et ◌̊). L'ordre des codes de caractères qui produit ce résultat graphique est *caractère consonne de base + caractère voyelle + caractère marque de ton*.

Figure 2-10. Interaction des caractères diacritiques

ã	LETTRE MINUSCULE LATINE A TILDE LETTRE MINUSCULE LATINE A + DIACRITIQUE TILDE
ȧ	LETTRE MINUSCULE LATINE A + DIACRITIQUE POINT EN CHEF
ậ	LETTRE MINUSCULE LATINE A TILDE + DIACRITIQUE POINT SOUSCRIT LETTRE MINUSCULE LATINE A + DIACRITIQUE TILDE + DIACRITIQUE POINT SOUSCRIT LETTRE MINUSCULE LATINE A + DIACRITIQUE POINT SOUSCRIT + DIACRITIQUE TILDE
ạ̇	LETTRE MINUSCULE LATINE A + DIACRITIQUE POINT EN CHEF + DIACRITIQUE POINT SOUSCRIT LETTRE MINUSCULE LATINE A + DIACRITIQUE POINT SOUSCRIT + DIACRITIQUE POINT EN CHEF
ấ	LETTRE MINUSCULE LATINE A ACCENT CIRCONFLEXE ET ACCENT AIGU LETTRE MINUSCULE LATINE A ACCENT CIRCONFLEXE + DIACRITIQUE ACCENT AIGU LETTRE MINUSCULE LATINE A + DIACRITIQUE ACCENT CIRCONFLEXE + DIACRITIQUE ACCENT AIGU
ấ	LETTRE MINUSCULE LATINE A ACCENT AIGU + DIACRITIQUE ACCENT CIRCONFLEXE LETTRE MINUSCULE LATINE A + DIACRITIQUE ACCENT AIGU + DIACRITIQUE ACCENT CIRCONFLEXE

L'ordre d'empilement implicite ne s'applique pas à certains diacritiques particuliers qui se placent de côté ou forment une ligature avec un diacritique adjacent (cf. *figure 2-11*). Lorsqu'ils se placent horizontalement, la direction prédominante de l'écriture dans laquelle les codes sont utilisés détermine leur ordre. Ainsi, dans le cas d'une écriture de gauche à droite, les accents horizontaux seraient codés de gauche à droite. L'exemple du haut de la *figure 2-11* respecte cette convention alors que celui du bas la contrevient.

Figure 2-11. Empilement horizontal irrégulier

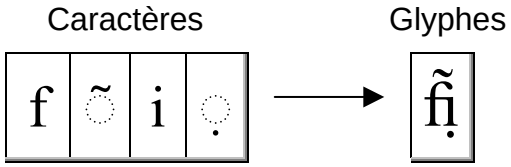
ᾴ	LETTRE MINUSCULE GRECQUE ALPHA + DIACRITIQUE VIRGULE EN CHEF (esprit doux, psili) + DIACRITIQUE ACCENT AIGU (oxia)	Empilement correct
ᾴ	LETTRE MINUSCULE GRECQUE ALPHA + DIACRITIQUE ACCENT AIGU (oxia) + DIACRITIQUE VIRGULE EN CHEF (esprit doux, psili)	Empilement incorrect

Les caractères qui dérogent de la sorte à l'empilement ordinaire appartiennent à des écritures ou à des alphabets particuliers. Ainsi en grec, lorsqu'un esprit, U+0313 ◌̓ DIACRITIQUE VIRGULE EN CHEF (esprit doux) ou 0314 ◌̔ DIACRITIQUE VIRGULE RÉFLÉCHIE EN CHEF (esprit rude), précède un accent aigu ou grave, les deux diacritiques doivent-ils s'écrire côte à côte au-dessus de la lettre de base. L'accent ne doit donc pas s'écrire au-dessus de l'esprit. Ici l'ordre des codes est *code du caractère de base + code de l'esprit + code de l'accent*. Cet exemple démontre que la disposition correcte des signes diacritiques dépend de l'écriture employée.

Caractères de base multiples

Quand les glyphes qui représentent deux caractères de base fusionnent pour former une ligature, il faut alors rendre correctement les caractères combinatoires par rapport au glyphe ligaturé (cf. *figure 2-12*). Au niveau interne, le logiciel doit donc distinguer les diacritiques qui s'appliquent à la première partie de la ligature de ceux qui s'appliquent à la seconde. Pour un examen des méthodes générales de positionnement des diacritiques, veuillez vous reporter à la section *5.13, Stratégies pour traiter les signes à chasse nulle*.

Figure 2-12. Caractères de base multiples



Dans la plupart des écritures, il est rare que l'on rencontre plusieurs caractères de base. Il arrive cependant, dans quelques écritures comme l'arabe, que cette situation se produise fréquemment quand on utilise les points-voyelles. Ce phénomène se produit de par le grand nombre de ligatures en arabe, où chaque élément d'une ligature est consonne et peut donc porter un point-voyelle. On rencontre même des ligatures de plus de deux consonnes, chacune de ces parties peut être voyellisée.

Clones des diacritiques européens avec chasse

Par convention, les diacritiques utilisés par le standard Unicode peuvent être affichés isolément en leur associant U+0020 ESPACE ou U+00A0 ESPACE INSÉCABLE. On peut se servir de cette astuce lorsque, par exemple, on désire parler du diacritique lui-même en tant que signe plutôt que de l'utiliser normalement dans le texte. Dans le but principal d'assurer une compatibilité avec les normes de jeux de caractères existantes, le standard Unicode code de nombreux clones de diacritiques européens fréquents ; ces clones chassent. Des renvois dans la liste des noms du chapitre *15, Tableaux de codes*, relient ces caractères apparentés.

2.7 Valeurs de non-caractères et de caractères spéciaux

Le standard Unicode comprend quelques caractères importants ayant un comportement particulier. Cette section en présente quelques-uns. Il importe que les mises en œuvre traitent ces caractères correctement. Pour une liste complète de ces caractères voir la section *3.9, Propriétés des caractères spéciaux* ; pour plus de renseignements au sujet de ces caractères voir les sections *14.1, Codes de commande*, *14.2, Commandes de disposition* et *14.6, Codes spéciaux.*

Indicateur d'ordre des octets (IOO)

La forme de stockage canonique d'un texte brut Unicode correspond à une suite de seizets sensibles à l'ordre des octets utilisé dans une éventuelle sérialisation du texte, par exemple lors de l'enregistrement de ce texte dans un fichier ou lors de sa transmission sur un réseau. Certains processeurs placent l'octet de moindre poids en position initiale, pour d'autres c'est l'octet le plus significatif qui occupe cette place. Idéalement, toutes les mises en œuvre du standard Unicode sérialiseraient les caractères d'une seule manière. Ceci signifierait toutefois qu'une des classes de processeurs se verrait forcer de permuter systématiquement les octets

à la lecture ou à l'écriture de fichiers de texte brut, même si ces fichiers ne devaient jamais quitter le système sur lequel ils ont été créés.

Afin d'indiquer de manière efficace l'ordre de sérialisation des octets dans un texte, le standard Unicode prévoit deux numéros, U+FEFF ESPACE INSÉCABLE SANS CHASSE (*indicateur d'ordre des octets*) et FFFE (pas un caractère), l'un étant l'image symétrique de l'autre en termes de sérialisation. L'indicateur d'ordre des octets (IOO) n'est pas un caractère de commande qui sélectionnerait l'ordre des octets du texte, sa fonction est plutôt de signaler l'ordre des octets utilisé dans un fichier.

Signature Unicode. Un IOO initial peut également servir à indiquer qu'un fichier contient du code Unicode. La suite $FE_{16}\,FF_{16}$ (ou sa contrepartie inversée, $FF_{16}\,FE_{16}$) est extrêmement rare au début d'un fichier texte qui utiliserait un autre codage de caractères. Il est donc peu probable que cette suite soit prise pour des données textuelles réelles, que les caractères de ce texte soient codés sur un ou plusieurs octets.

Les flux de données (ou les fichiers) qui commencent par U+FEFF *indicateur d'ordre des octets* contiennent donc probablement des valeurs Unicode. On recommande aux applications qui envoient ou reçoivent des flux de caractères codés non typés d'utiliser cette signature. Si on utilise d'autres méthodes de signalisation, on s'abstiendra d'employer les signatures Unicode.

Pour se conformer au standard, il n'est pas nécessaire d'utiliser un IOO comme signature. Veuillez vous référer la section *14.6, Codes spéciaux* pour un supplément d'information sur l'indicateur d'ordre des octets et son emploi en tant que signature de codage.

Valeurs de non-caractères spéciaux

U+FFFF et U+FFFE. Ces valeurs de code ne représentent pas des caractères Unicode. On réserve U+FFFF à l'utilisation privée des programmes, comme sentinelle ou tout autre signal. Remarquez que U+FFFF est la représentation sur 16 bits de –1 dans la notation complément à deux. Les programmes qui reçoivent ce code ne sont pas obligés de l'interpréter. Cependant, les règles de l'art veulent qu'on note que ce code ne correspond à aucun caractère et de prendre les mesures qui s'imposent en indiquant, par exemple, la corruption probable du texte. U+FFFE est similaire à tous égards à U+FFFF, si ce n'est qu'il est l'image symétrique de U+FEFF ESPACE INSÉCABLE SANS CHASSE (indicateur d'ordre des octets). La présence de U+FEFF constitue un indice sérieux d'inversion des octets du texte en question. Voir la section *14.6, Codes spéciaux*.

Séparateurs

Séparateurs de lignes et de paragraphes. Pour des raisons historiques, différentes plates-formes utilisent de manière différente les retours de chariot et les passages à la ligne. Le standard Unicode prévoit (et encourage l'utilisation) des caractères séparateurs de lignes et de paragraphes pour préciser ces limites. L'algorithme bidirectionnel rend nécessaire la séparation des textes en paragraphes (cf. chapitre *3, Conformité*). Étant donné que ces entités sont des codes séparateurs, il est inutile que ces entités précèdent la première ligne ou le premier paragraphe ou suivent la dernière ligne ou le dernier paragraphe. Cf. également la section *14.2, Commandes de disposition*.

Interprétation des CR/LF. Le standard Unicode n'attribue pas de sens particulier aux caractères U+000D RETOUR DE CHARIOT (CR) et U+000A PASSAGE À LA LIGNE (LF). Ces codes existent afin de représenter n'importe quel retour de chariot (CR) ou passage à la ligne (LF) employés

par un protocole de niveau supérieur ou conservés dans un texte traduit d'une autre norme. Chaque application est responsable d'interpréter ces codes, de décider s'ils doivent être utilisés ou non et si ces caractères doivent être utilisés en paires (CR/LF) ou si un seul des codes suffit. Voir également la section *5.9, Traitement des lignes* et le rapport technique Unicode n°13 *Unicode Newline Guidelines* présent sur le disque accompagnant ce livre ou sur le site Internet Unicode pour une version tenue à jour.

Caractères de formatage et de disposition

Le standard Unicode définit plusieurs caractères qui permettent de préciser la liaison cursive des caractères, l'ordre bidirectionnel et des formes supplémentaires d'affichage. Il est explicitement spécifié que ces caractères n'affectent pas la coupure de ligne. Contrairement aux espaces et aux autres délimiteurs, ils ne servent pas à indiquer les limites d'un mot, d'une ligne ou d'une autre unité. La section *14.2, Commandes de disposition,* précise leur utilisation en ce qui a trait à la composition et au formatage.

Caractère de remplacement

U+FFFD CARACTÈRE DE REMPLACEMENT correspond au caractère de substitution générique du standard Unicode. Il peut servir à remplacer n'importe quel caractère « inconnu » provenant d'un autre codage et qui ne peut être converti à une valeur Unicode connue. Voir les sections *5.3, Caractères inconnus ou manquants*, et *14.6, Codes spéciaux*.

2.8 Commandes et suites de commandes

Caractères de commande

Le standard Unicode prévoit 65 numéros pour représenter les caractères de commande. Ils sont situés dans les intervalles U+0000..U+001F et U+007F..U+009F. Ils correspondent aux octets de commande 00_{16} à $1F_{16}$ (commandes C0) et $7F_{16}$ à $9F_{16}$ (*suppression* (*del*) et commandes C1). Ainsi, si la version à 8 bits de la tabulation horizontale est située à la position 09_{16} alors le code Unicode *tab* est situé au U+0009. Pour convertir en UTF-16 des codes de commande présents dans des textes à 8 bits existants, il suffit donc de les étendre à des entités à 16 bits en y ajoutant 8 bits mis à zéro.

Les programmes qui se conforment au standard Unicode peuvent traiter ces caractères de commande exactement de la même manière que leurs équivalents à 7 ou 8 bits dans d'autres protocoles, comme l'ISO/CEI 2022 et l'ISO/CEI 6429. L'utilisation de ces caractères constitue un protocole de niveau supérieur et déborde donc le cadre du standard Unicode. De même, l'utilisation des séquences de commande de l'ISO/CEI 6429 :1992 pour préciser le formatage bidirectionnel constitue un protocole de haut niveau légitime ajouté par-dessus le texte brut Unicode. Comme pour tous les protocoles de niveau supérieur, il faut que l'émetteur et le récepteur se soient au préalable entendus sur le protocole commun adopté.

Séquences d'échappement. Lors de la conversion vers Unicode d'un texte qui comprend des séquences d'échappement, on doit convertir le texte vers les caractères Unicode équivalents[3]. La conversion des séquences d'échappement vers Unicode, caractère par caractère (ESC-A devient ainsi U+001B ÉCHAPPEMENT suivi de U+0041 A LETTRE , MAJUSCULE LATINE A) permet

[3] Il faut donc s'abstenir d'interpréter les séquences d'échappement (ISO 2022, par exemple) et ainsi obtenir d'autres caractères.

d'effectuer une conversion inverse sans que le programme de conversion ne soit obligé d'interpréter les séquences d'échappement elles-mêmes.

Séquences de commandes précisant des propriétés supplémentaires du texte. Si un système utilise dans un texte des séquences de codes de commande afin d'y imbriquer des informations supplémentaires relatives à ce texte (telles que des attributs de formatage) alors ces séquences forment un protocole de niveau supérieur qui dépasse le cadre du standard Unicode. Ces protocoles de haut niveau ne sont pas précisés par le standard Unicode, on ne peut présumer de leur existence sans un accord entre les parties qui s'échangent ces données.

Représentation des suites de commandes

L'architecture de codage Unicode permet de représenter des séquences de commandes mais il faut alors les représenter selon la forme de stockage choisie (par ex. en termes de seizets en UTF-16). Ainsi, supposons qu'une application permette l'incorporation d'informations relatives aux polices à utiliser grâce à des séquences d'octets. Ci-dessous, **^A** représente le code de commande C0 « 01_{16} », **^B** représente le code de commande « 02_{16} » et ainsi de suite :

^ATimes**^B** = **01**,54,69,6D,65,73,**02**

La séquence Unicode correspondante serait :

^ATimes**^B** = **0001**,0054,0069,006D,0065,0073,**0002**

En d'autres mots, chaque unité de stockage UTF-16 est un seizet dont les 8 premiers bits sont à zéro et les derniers 8 correspondent à l'octet original.

Dans le cas où les données imbriquées ne sont pas interprétées comme une suite de caractère par le protocole, on pourra alors coder l'information de la manière suivante :

^ATimes**^B** = **0001**,5469,6D65,7300,**0002**

On ne pourra jamais coder l'information de cette manière-ci en UTF-16

^ATimes**^B** = **01**54,696D,6573,**02**00

car, en UTF-16, cette suite représente quatre caractères codés — 0154 Ŕ LETTRE MAJUSCULE LATINE R ACCENT AIGU, deux caractères han (U+696D 業 et U+6573 敳) et enfin U+0200 Ȁ LETTRE MAJUSCULE LATINE A DOUBLE ACCENT GRAVE. Aucun de ces caractères n'est un caractère de commande. Si une séquence de commande comprend des données binaires imbriquées, il n'est pas nécessaire d'étendre les octets en seizets en y ajoutant des zéros puisque la séquence de commande constitue le protocole de niveau supérieur. Cependant, l'ajout de zéros permet aux algorithmes de conversion de code de réussir même en ne connaissant pas explicitement les séquences de commande utilisées.

2.9 Conformité au standard Unicode

Le Chapitre *3, Conformité,* précise l'ensemble des critères non ambigus qu'une mise en œuvre conforme à Unicode doit respecter afin de pouvoir communiquer avec d'autres mises en œuvre conformes. La section ci-dessous fournit quelques exemples de conformité et de non-conformité qui complètent la description formelle de conformité.

Une mise en œuvre conforme au standard Unicode aura les caractéristiques suivantes :

- Elle traite les caractères en tant qu'unités de 16 bits[4] .

  U+2020 (c'est-à-dire 2020_{16}) est le numéro de caractère Unicode de l'OBÈLE « † » et non de deux espaces ASCII.

- Elle identifie et interprète les caractères selon les propriétés et règles de ce standard qui s'y rapportent.

  U+2423 représente « ␣ » BOÎTE OUVERTE et non ゐ HIRAGANA I MINUSCULE (le caractère abstrait qui correspond aux octets 2423_{16} en JIS).

  U+00F4 « ô » est équivalent à U+006F « o » suivi de U+0302 « ◌̂ », mais pas à U+0302 suivi de U+006F.

  U+05D0 « א » suivi de U+05D1 « ב » ressemble, à l'affichage, à « אב » et non à « בא ».

  Quand une mise en œuvre prend en charge les caractères arabes ou hébreux et qu'elle affiche ces caractères, ils doivent être affichés dans l'ordre défini par l'algorithme bidirectionnel décrit à la section *3.12, Comportement bidirectionnel*.

  Quand une mise en œuvre prend en charge les caractères arabes, dévanâgarî, tamouls ou d'autres écritures aux glyphes contextuels et les affiche, il faut alors au moins que les caractères soient contextualisés selon les informations fournies aux sections *9.2, Arabe*, *10.1, Dévanâgarî*, et *10.6, Tamoul*. On peut utiliser des règles de formage plus complexes le cas échéant.

- Elle n'utilise pas de code non affecté.

  U+2073 n'est pas attribué et on ne peut donc l'utiliser pour coder « ^{3} » (exposant 3) ou tout autre caractère.

- Elle ne corrompt pas les caractères inconnus.

  U+2029 est le SÉPARATEUR DE PARAGRAPHES, les applications qui ne le prennent pas en charge ne doivent pas le supprimer.

  Il ne faut pas changer U+03A1 « P » LETTRE MAJUSCULE GRECQUE RHÔ en U+00A1 (premier octet supprimé), U+0050 (transformé en un P latin), U+A103 (octets inversés) ou tout autre code qui ne serait U+03A1.

Toutefois, il est permis aux mises en œuvre conformes :

- De ne prendre en charge qu'un sous-ensemble de caractères Unicode.

  Une application peut ne pas fournir de symboles mathématiques ou ne pas traiter l'écriture thaïe.

- De transformer les données à dessein.

  Conversion vers les majuscules : « a » transformé en « A »

  Romaji vers kana : « kyo » transformé en きょ

[4] Selon nous, cette exigence n'a plus cours dans Unicode 3.1. Il s'agit sans doute d'un reliquat du vieux slogan « Unicode = ASCII 16 bits ». Dans Unicode 1.0, il n'y avait ni UTF-8 ni seizets d'indirection, on pouvait donc affirmer qu'un caractère égalait un seizet. Cette affirmation nous est fausse à l'heure actuelle en UTF-16, UTF-32 tout comme en UTF-8.

U+247D « ⑽ » décomposé en 0028 0031 0030 0029

- D'utiliser Unicode pour bâtir un protocole de niveau supérieur.

  Compression des caractères

  Utilisation de formats de fichier à texte enrichi

- De définir des caractères dans la zone à usage privé.

  Comme exemples de caractères qui peuvent être codés dans la zone à usage privé, on peut citer les caractères idéographiques supplémentaires (les *gaiji*) ou les logotypes d'entreprise.

- De ne pas prendre en charge l'algorithme bidirectionnel, la contextualisation des caractères pour les mises en œuvre qui ne prennent pas en charge les écritures complexes, comme l'arabe ou la dévanâgarî.

- De ne pas prendre en charge l'algorithme bidirectionnel ou la contextualisation des caractères dans des applications qui n'affichent ou n'impriment pas de caractères, comme des serveurs ou des programmes qui ne font qu'analyser ou transcoder du texte, comme c'est le cas pour un processeur XML.

On dira qu'une conversion de code à partir d'autres normes vers Unicode est conforme si le tableau de conversion produit des conversions exactes dans les deux sens.

Caractères non utilisés dans un sous-ensemble

Pour qu'elle soit conforme, le standard Unicode n'exige pas d'une application qu'elle soit capable d'interpréter ou d'afficher tous les caractères Unicode. Beaucoup de systèmes auront à leur disposition des polices pour certaines écritures du standard mais non d'autres. Le tri ou d'autres manipulations de texte pourront aussi être mis en œuvre pour un nombre restreint de langues. En conséquence, une mise en œuvre peut n'interpréter qu'un sous-ensemble de caractères.

Le standard Unicode ne prévoit aucun mécanisme formel pour identifier ce sous-ensemble. En outre, ce sous-ensemble varie généralement selon les aspects de la même mise en œuvre. Ainsi, une application peut être capable de lire, écrire ou stocker tous les caractères, et de trier un certain sous-ensemble selon les règles d'une ou plusieurs langues (et le reste arbitrairement), mais n'avoir accès qu'à des polices correspondant à une seule langue. La même application pourrait être capable de rendre correctement des écritures supplémentaires dès l'installation des polices appropriées. C'est pourquoi le sous-ensemble des caractères interprétables n'est d'ordinaire pas un concept statique.

La conformité au standard Unicode *implique* que, lorsqu'un texte prétend être non modifié, les caractères non interprétables ne soient ni ôtés ni modifiés (cf. également la section *3.1, Exigences de conformité*).

2.10 Référence aux versions du standard Unicode

Pour la plupart des codages, le répertoire est fermé (et souvent de petite taille). Une fois établi, un tel répertoire ne change plus jamais. On considère l'ajout de nouveaux caractères à un répertoire fermé comme la création d'un nouveau répertoire à qui l'on attribuera un nouveau numéro de catalogue et qui constituera ainsi un nouvel objet.

Le répertoire d'Unicode, par contre, est intrinsèquement ouvert. Unicode étant un codage universel, tout caractère abstrait qui pourrait jamais être codé est un membre potentiel du jeu de caractères, que ce caractère soit actuellement connu ou non.

Chaque nouvelle version du standard Unicode remplace la dernière et la rend périmée, mais les mises en œuvre — et, de manière plus fondamentale, les données — ne sont pas mises à jour instantanément. En général, les changements apportés par les versions majeures et mineures introduisent de nouveaux caractères, ceux-ci ne causent pas de problèmes particuliers pour ce qui à trait aux données préexistantes. Le Comité technique Unicode n'enlèvera ni ne déplacera de caractères, mais il pourra décréter que certains devraient être évités. Cette démarche ne les élimine donc ni du standard ni des données existantes. Les numéros de caractère ne seront donc jamais utilisés pour désigner d'autres caractères, mais on découragera à l'avenir fortement leur utilisation.

Les mises en œuvre devraient autant que possible être compatibles avec les versions futures d'Unicode. En d'autres mots, elles devraient accepter des textes qui pourraient être exprimés à l'aide de futures versions du standard Unicode et s'attendre à ce que de nouveaux caractères puissent apparaître dans ces versions. Elles devraient donc traiter les numéros non attribués de la même manière que des caractères non supportés. Cf. *5.3, Caractères inconnus ou manquants.*

Un changement de version peut également impliquer des changements aux propriétés des caractères existants. Ceci se traduit par des modifications au fichier UnicodeData.txt ainsi qu'à d'autres fichiers connexes ; ces mises à jour sont publiées conjointement à la sortie de la nouvelle version du standard. Des changements aux fichiers de propriétés peuvent entraîner une modification du comportement des programmes qui s'en servent.

Numérotation des versions. Les numéros de version du standard se composent de trois champs : le numéro de version majeure, le numéro de version mineure et le numéro de mise à jour. Les différences entre ces numéros sont expliquées ci-dessous :

- majeur — additions importantes au standard, publié sous la forme d'un livre.
- mineur — ajout de caractères ou certains changements normatifs plus conséquents, publiés sous la forme d'un rapport technique sur le site internet Unicode.
- mise à jour — tout autre changement aux sections normatives ou à d'importantes sections informatives du standard qui pourrait influencer le comportement des mises en œuvre. Ces changements entraînent la production d'un nouveau fichier UnicodeData.txt et d'autres fichiers connexes de la *Base de données de caractères Unicode*.

Pour plus des renseignements sur la version actuelle ou sur les anciennes versions du standard Unicode, veuillez consulter <http://www.unicode.org/unicode/standard/versions/> sur le site internet du consortium Unicode.