Chapitre 3

Conformité

Ce chapitre définit la conformité au standard Unicode en fonction des principes et de l'architecture de codage qui le sous-tendent. La première section est constituée des clauses de conformité, suivent des sections qui définissent plus précisément les termes techniques utilisés dans ces clauses. Les dernières sections reprennent les algorithmes officiels qui font partie des exigences de conformité et auxquels les clauses de conformité font référence. Ces algorithmes spécifient les résultats prescrits plutôt que leur mise en œuvre précise ; sont donc conformes toutes les mises en œuvre qui produisent des résultats identiques à ceux de ces algorithmes officiels.

Dans ce chapitre, la lettre C désigne les sous-sections de conformité. La lettre D précède les définitions. Les listes à puce correspondent à des commentaires relatifs aux définitions ou aux sous-alinéas.

Sauf dans le cas de la section *3.12, Comportement bidirectionnel*, la numération des règles et des définitions correspond à celle de la version 2.0 du standard Unicode. Quand de nouvelles règles ou définitions se sont ajoutées, on a fait suivre les numéros d'une lettre afin de conserver la numérotation originelle, par exemple : *D7a*.

3.1 Exigences de conformité

Cette section précise les exigences officielles de conformité pour tout processus qui met en œuvre la version 3.0.1 du standard Unicode. Il convient de remarquer que cette section a été revue depuis la dernière version du standard Unicode. Ces révisions ne changent pas en substance les exigences de conformité prescrites auparavant, mais elles les formalisent et les étendent afin de permettre l'utilisation de formats transformés. Les mises en œuvre qui vérifiaient les conditions de conformité des versions précédentes du standard Unicode, se conformeront également aux clauses révisées.

Ordre des octets

C1 *Un processus doit interpréter les valeurs Unicode[1] en tant que quantités à 16[2] bits (des seizets).*

- Les valeurs Unicode peuvent être stockées dans des mots-machines à 16 bits.
- Pour des renseignements sur l'utilisation de `wchar_t` ou d'autres types de données dans les langages informatiques qui permettent de représenter les valeurs Unicode, se rapporter à la section *5.3, wchar_t ANSI/ISO C*.

C2 *Le standard Unicode ne précise pas l'ordre des octets au sein des unités de stockage.*

[1] Valeurs Unicode n'est défini nulle part. Ce terme apparaît pour la première fois dans le dernier paragraphe de la section 2.7. Le modèle de codage (UTR17) n'en parle pas. Selon le cas, il correspond à « caractère Unicode » ou « numéro de caractère ».

[2] Cette clause est un reliquat de l'époque de l'ASCII à 16 bits, elle semble avoir échappé aux révisions des versions 2.0 et 3.0 . Cette clause ne vaut qu'en UTF-16.

- Certains ordinateurs ordonnent les octets en mémoire en privilégiant l'octet de poids le plus fort, d'autres préféreront stocker d'abord l'octet de poids le plus faible. On nomme respectivement ces deux méthodes de stockage *grand-boutienne* et *petit-boutienne*.

C3 En absence de protocole de niveau supérieur, un processus doit interpréter unité de stockage qui a été sérialisée en octets en commençant par l'octet de poids le plus fort.

- La majorité des échanges se produit entre processus qui s'exécutent sur une même machine ou des machines de même architecture. L'échange de textes Unicode au sein d'un même domaine selon un ordre des octets particulier à ce domaine est donc parfaitement conforme. Ceci limite donc le rôle de l'ordre canonique des octets à l'échange de textes Unicode au-delà d'un domaine défini ou lorsque la nature du domaine expéditeur est inconnue. Pour un examen de l'indicateur d'ordre des octets (IOO) comme signal de l'ordre de sérialisation des octets, se référer à la section *2.7, Valeurs de non-caractères et de caractères spéciaux*.

Valeurs de codes illégales

C4 Un processus ne peut interpréter comme caractère un seizet d'indirection non apparié.

C5 Un processus ne peut interpréter un non-caractère comme caractère abstrait.

- Les programmes peuvent utiliser ces numéros comme valeurs sentinelles ou délimiteurs, ils ne peuvent toutefois jamais être exportés publiquement.

C6 Un processus ne peut interpréter aucun caractère non attribué comme caractère abstrait.

- Ces clauses n'excluent pas l'attribution d'une sémantique générique (par exemple, l'affichage d'un glyphe qui indiquerait le bloc de caractères) qui permette de traiter de manière élégante la présence de unités de stockage correspondant à des sous-ensembles non supportés ou à des seizets d'indirection non appariés.

- Les numéros de caractère des zones à usage privé sont attribués, mais les processus conformes peuvent leur attribuer le sens qu'ils désirent.

Interprétation

C7 Si un processus interprète la représentation codée d'un caractère, il doit le faire selon la sémantique des caractères établie par ce standard.

- Cette restriction n'interdit pas à un processus de transformer à l'interne des caractères qui ne sont jamais visibles à l'extérieur du processus en question.

C8 Un processus ne doit pas supposer qu'il doive interpréter une représentation codée de caractère particulière.

- Ce standard ne précise pas de moyen qui permette de préciser un sous-ensemble de caractères à interpréter.

- La signification d'une valeur de code dans la zone à usage privé ne relève pas de ce standard.

- Bien que ces clauses ne visent pas à interdire l'énumération ou la spécification des caractères qu'un processus ou un système sont capables d'interpréter, elles forcent à

distinguer énumération des sous-ensembles supportés et conformité. En pratique, tout système peut de temps en temps recevoir des codes de caractère inconnu qu'il ne peut interpréter.

C9 Un processus ne peut supposer que deux suites de caractères canoniquement équivalentes sont interprétées de manières distinctes[3]*.*

- En principe, une mise en œuvre doit toujours interpréter identiquement deux suites de caractères canoniquement équivalentes. Il existe des raisons pratiques où des mises en œuvre peuvent à juste titre les distinguer.

- Même les processus qui ne distinguent pas habituellement les suites de caractères canoniquement équivalentes peuvent adopter, à bon escient, un comportement irrégulier. Parmi ceux-ci, les substitutions de repli effectuées par les processus qui ne prennent pas en charge le positionnement correct des diacritiques ou les modes « montrer le texte caché » qui révèlent la structure de la représentation en mémoire ou, enfin, le choix d'ignorer dans les tris lexicographiques les suites combinatoires qui ne font pas partie du répertoire de la langue précisée (cf. la section *5.13, Stratégies pour traiter les signes à chasse nulle*).

Modification

C10 Un processus qui prétend ne pas modifier le sens d'une représentation codée valable de caractères ne peut la modifier sauf pour remplacer, le cas échéant, des suites de caractères par des suites canoniquement équivalentes ou pour supprimer les non-caractères[4]*.*

- Si, lors d'un traitement, un non-caractère survient et que celui-ci n'a pas de rôle particulier interne au programme, une mise en œuvre peut signaler une erreur, supprimer ou ignorer ce non-caractère. Si on ne choisit aucune de ces options, il faut traiter le non-caractère comme un point de code non affecté.
- Le remplacement d'une suite de caractères par une suite équivalente de compatibilité *change* l'interprétation du texte.
- Le remplacement ou la suppression d'une suite de caractères qu'un processus ne traite pas ou ne peut traiter *change* l'interprétation du texte.
- Changer l'ordre des bits ou des octets lors d'une transformation entre des machines à l'architecture différente *ne modifie pas* l'interprétation du texte.
- Transformer un texte en une autre forme de codage *ne modifie pas* l'interprétation d'un texte.

Transformations

C11 Quand un processus interprète une suite d'octets dans un format transformé d'Unicode (UTF), il doit interpréter la suite d'octets conformément à la sémantique établie par ce standard pour la séquence de caractères Unicode correspondante.

C12 (a) Quand un processus produit des données dans un format transformé d'Unicode, il ne doit pas transmettre de suites d'unités de stockage mal formées.

---

[3] En d'autres mots, même si un processus a le droit de distinguer deux suites canoniquement équivalentes, il ne peut imposer à un autre processus de faire cette même distinction. Cette clause maintient solidement l'équivalence canonique, tout en donnant juste ce qu'il faut de corde aux mises en œuvre pour s'en tirer.

[4] Par exemple, le remplacement de « e accent aigu » (une suite d'un seul caractère) par « e » + « accent aigu » (suite équivalente).

(b) Quand un processus interprète des données en format transformé d'Unicode, il doit traiter une suite illégale d'unités de stockage comme une erreur.

(c) Un processus conforme ne doit pas interpréter comme caractères une suite illégale d'unités de stockage UTF[5]

(d) Aucune information ne doit être codée à l'aide de suites d'unités de stockage irrégulières[6] *UTF.*

Les clauses b et c ainsi que les notes ci-dessous ont été ajoutées à la version 3.0.1 pour des raisons de sécurité. Car, si l'ancienne version de cette clause interdisait la génération des formes UTF-8 non minimales et l'interprétation des suites illégales, elle n'interdisait pas l'interprétation des formes non minimales. Des problèmes de sécurité pourraient surgir si un programme venait à interpréter les formes non minimales :

1. Un processus A effectue des contrôles de sécurité mais ne vérifie pas la présence de formes non minimales.
2. Un processus B accepte la suite d'octets provenant du processus A et la transforme en UTF-16 tout en interprétant les formes non minimales.
3. Un texte UTF-16 peut alors comprendre des caractères qui auraient dû être éliminés par le processus A.

- La définition de chaque format transformé d'Unicode (UTF) précise les unités de stockage illégales pour ce format. Ainsi la définition d'UTF-8 (D36) énonce que les suites d'unités de stockage de type <C0, AF> sont illégales.

- On peut utiliser des fonctions internes qui ne vérifie pas systématiquement la présence de suites d'unités de stockage illégales. Toutefois, un processus conforme ne peut utiliser de telles fonctions qu'après s'être assuré qu'aucune suite illégale ne leur est fournie.

- Les processus qui exigent une représentation unique des caractères ne peuvent pas considérer les suites d'unités de stockage irrégulières comme des caractères. Ils peuvent, par contre, les éliminer ou les rejeter.

- Il est permis aux processus conformes de transformer les suites d'unités de stockage irrégulières en suites d'unités de stockage bien formées correspondantes.

- Les processus conformes ne peuvent interpréter les suites d'unités de stockage illégales. Toutefois, les clauses de conformité n'interdisent pas, par exemple, que les programmes utilitaires traitent du texte « tailladé ». Ainsi, un fichier UTF-8 pourrait contenir des suites de CRLF insérées tous les 80 octets par un mauvais logiciel de courrier. Il se pourrait donc que certaines suites UTF-* soient entrecoupées de CRLF, produisant de la sorte des suites illégales d'octets. Ce texte « tailladé » n'est plus de l'UTF-8. Les programmes conformes peuvent restaurer de tels textes en utilisant le fait que l'original était constitué d'une suite bien formée d'octets UTF-8. Ce type de restauration de données « tailladées » constitue cependant un cas particulier, on ne doit pas y avoir recours si cela pouvait occasionner des problèmes de sécurité.

[5] V. clause D31.

[6] V. clause D32.

Texte bidirectionnel

C13 *Un processus qui affiche un texte comprenant des caractères écrits de droite à gauche ou des codes d'enchâssement supportés par cette mise en oeuvre doit afficher, en l'absence de protocole de niveau supérieur, toutes les représentations visibles des caractères (à l'exception des caractères de formatage) dans le même ordre que celui que l'algorithme bidirectionnel produirait sur ce même texte.*

Rapports techniques Unicode

Les rapports techniques suivants ont été approuvés et sont considérés comme faisant partie intégrale de la version 3.0 du standard Unicode. Ces rapports contiennent des textes normatifs ou informatifs, ou les deux. Toute référence à la version 3.0 du standard inclut automatiquement ces rapports techniques :

- RTU n° 11 : East Asian Width, version 5.0 ;
- RTU n° 13 : Unicode Newline Guidelines, version 5.0 ;
- RTU n° 14 : Line Breaking Properties, version 6.0 ;
- RTU n° 15 : Unicode Normalization Forms, version 18.0[7].

3.2 Sémantique

Cette section ainsi que les suivantes définissent plus précisément les termes utilisés dans les clauses de conformité.

D1 *Propriétés et comportement normatifs.* Voici la liste des propriétés normatives de caractère ainsi que les comportements normatifs du standard Unicode :

1. Propriétés simples ;
2. Combinaison de caractères ;
3. Décomposition canonique ;
4. Décomposition de compatibilité ;
5. Propriété d'indirection ;
6. Mise en ordre canonique ;
7. Comportement bidirectionnel, conformément à l'algorithme bidirectionnel d'Unicode ;
8. Comportement des jamos jointifs, conformément à la section *3.11, Comportement des jamos jointifs.*
9.

D2 *Sémantique des caractères :* la sémantique d'un caractère dépend de son nom, de son glyphe représentatif, de ses propriétés et comportement normatifs.

- Un caractère peut avoir une utilisation plus étendue que son nom ne le laisse supposer. Il faut interpréter la représentation codée, le nom et le glyphe représentatif

[7] Ce rapport correspond au chapitre 6 du présent ouvrage.

en contexte pour établir la sémantique d'un caractère. Ainsi, U+002E . POINT peut-il représenter le point final d'une phrase, un point d'abréviation, un séparateur de milliers en français[8] ou un séparateur décimal en anglais, et ainsi de suite.

- Il n'est pas nécessaire qu'un glyphe d'une application conforme soit identique au glyphe représentatif de la norme ou qu'il ait même un dessin similaire à celui-ci, il suffit simplement qu'on admette que les deux images sont des variantes d'un même caractère. Représenter le caractère 0061 a LETTRE MINUSCULE LATINE A par le glyphe « X » violerait son identité de caractère.

- Des protocoles de niveau supérieur peuvent prévaloir sur certains comportements normatifs implicites. Cependant, en absence de tels protocoles, le comportement normatif doit être respecté afin de conserver la sémantique des caractères.

- Les protocoles de niveau supérieur ne peuvent modifier ni les propriétés de combinaison de caractères, ni la mise en ordre canonique.

3.3 Caractères et représentations codées

D3 *Caractère abstrait :* unité d'information utilisée pour organiser, commander ou représenter des données textuelles.

- Quand il s'agit de représenter des données, celles-ci sont généralement des symboles et non des nombres, des sons ou des graphiques. Parmi ces données symboliques, on retrouve des lettres, des idéogrammes, des chiffres, des signes de ponctuation, des symboles techniques et divers signes du casseau.

- Un caractère abstrait n'a pas de forme concrète, il ne faut pas le confondre avec un glyphe.

- Un caractère abstrait ne correspond pas nécessairement à ce que l'utilisateur imagine être un « caractère », il ne doit pas non plus être confondu avec un *graphème*.

- Les caractères abstraits que le standard Unicode code sont appelés des caractères abstraits Unicode.

- Les caractères abstraits qui ne sont pas directement codés par le standard Unicode peuvent souvent être représentés à l'aide de suites de caractères combinatoires.

D4 *Suite de caractères abstraits :* séquence ordonnée de caractères abstraits.

Chaque caractère Unicode est codé au moins une fois[9]. Chaque correspondance entre un caractère abstrait et son numéro (sa valeur scalaire, cf. D28) se nomme un caractère codé. Chaque numéro de caractère peut être représenté soit par une seule valeur de code, soit par une suite de deux valeurs de code d'indirection[10].

[8] Bien que ceci ne soit plus le séparateur de milliers préconisé en France.

[9] Certains caractères abstraits ont plusieurs numéros dans Unicode, par exemple les équivalents de compatibilité ou même de (rares) doublons (dans le cas du coréen, ils sont reproduits sciemment à des fins de compatibilité).

[10] Ce commentaire constitue sans doute un reliquat de l'époque où tous les caractères Unicode étaient codés sur seize bits. Il vaudrait mieux dire, en accord avec la section *2.3, Modèle de caractères,* que chaque valeur scalaire Unicode peut être représentée, selon le format de stockage, sous la forme d'une suite d'octets (UTF-8), d'un seizet normal ou de deux seizets d'indirection (UTF-16) ou d'une valeur de 32 bits (UTF-32).

D5 *Unité de stockage :* la combinaison binaire minimale qui représente une unité de texte codé destinée à être traitée ou échangée.

- Les autres normes de codage de caractères utilisent habituellement des unités de stockage définies sous la forme d'octets. C'est le cas pour le format transformé du standard Unicode UTF-8. Cependant, les unités de stockage utilisées dans la forme UTF-16 du standard Unicode sont des seizets.

- L'industrie informatique nomme parfois les *unités de stockage* des *unités de code*.

- Un seul caractère abstrait peut correspondre à un ou plusieurs caractères codés ; exemples : 00C5 Å LETTRE MAJUSCULE LATINE A ROND EN CHEF et 212B Å SYMBOLE ANGSTRÖM.

- Plusieurs unités de stockage peuvent être nécessaires pour représenter un seul caractère abstrait. Ainsi, l'octet correspond-il à l'unité de stockage en SJIS : une seule unité représente des caractères comme le « a » alors qu'il en faut deux pour représenter les idéogrammes.

- Pour certains codages, les unités de stockage ne peuvent s'utiliser indépendamment les unes des autres pour représenter un caractère codé. C'est le cas des seizets d'indirection isolés (supérieurs ou inférieurs) en UTF-16, les octets 80-FF en UTF-8 ainsi que les octets 81-9F, E0-EF en SJIS.

D6 *Représentation codée de caractères :* suite ordonnée d'une ou plusieurs unités de stockage associées à un caractère codé[13] dans un répertoire de caractères donné.

- En UTF-16, une unité de stockage code généralement un caractère abstrait Unicode; la seule exception comprend les paires de seizets d'indirection qui constituent un mécanisme d'extension pour coder des caractères sur d'autres plans que le PMB.

D7 *Suite de caractères codés* : suite ordonnée de représentations codées de caractères.

Le texte du standard Unicode utilise habituellement le terme *caractère* pour désigner une représentation codée de caractères sauf en cas d'équivoque possible. De même, le terme *suite de caractères* désigne-t-il habituellement une suite de caractères codés.

D7a *Caractère à éviter :* caractère codé dont l'emploi est fortement découragé. Le standard conserve ces caractères mais il ne faut pas les utiliser.

- Le standard conserve les caractères à éviter afin que les données conformes préexistantes demeurent conformes dans les prochaines versions du standard. Il faut distinguer les caractères à éviter des caractères désuets.

- Les caractères désuets sont d'anciens caractères, ils ne se rencontrent pas dans les textes modernes mais ils ne sont pas à éviter ; on ne décourage pas leur utilisation.

D7b *Non-caractère :* un numéro définitivement réservé à un usage interne et qui ne doit jamais être échangé. Pour Unicode 3.0, il s'agit des valeurs U+*n*FFFE et U+*n*FFFF, où *n* vaut 0 à 10_{16}.

---

[12] À noter qu'ici le texte parle de caractères et non plus d'unités de stockage.

[13] Caractère codé plutôt qu'abstrait car un caractère abstrait peut avoir plusieurs numéros, la représentation codée (qui représente la forme stockée) doit donc être associée à chaque numéro.

- Pour plus de renseignements, veuillez vous référer aux paragraphes « Valeurs de non-caractères spéciaux » à la section 2.7, *Valeurs de non-caractères et de caractères spéciaux*, et « Non-caractères » à la section *14.6, Codes spéciaux.*

- Ces numéros sont définitivement réservés comme non-caractères, il se peut cependant qu'à l'avenir de nouveaux numéros soient attribués pour désigner d'autres non-caractères.

D8 *Protocole de niveau supérieur :* tout accord sur l'interprétation de caractères Unicode qui dépasse le cadre de cette norme. Cet accord ne doit pas nécessairement être annoncé de manière formelle et explicite dans les données échangées, il peut être implicite à son utilisation dans un contexte donné.

3.4 Propriétés simples

La version 3.0 du standard Unicode définit pour les caractères les propriétés normatives simples suivantes : *casse*, *valeur numérique*, *directionalité* et *miroir*. Le chapitre *4, Propriétés des caractères,* recense ces propriétés ainsi que les caractères qui leur correspondent. Ces correspondances représentent les propriétés implicites que les processus conformes doivent assigner aux caractères, en l'absence de protocole supérieur explicite dérogatoire. Des propriétés supplémentaires propres à certains caractères (comme la définition et l'utilisation des caractères de forçage droite-à-gauche ou des espaces sans chasse) font l'objet d'un examen dans les sections appropriées de ce standard.

- La *Base de données de caractères Unicode* reprend des propriétés supplémentaires, comme les correspondances de catégorie ou de casse, celles-ci sont informatives plutôt que normatives.

- L'interprétation de certaines propriétés (comme la casse) est indépendante du contexte alors que l'interprétation d'autres propriétés (comme la directionalité) ne s'applique qu'à une suite de caractères dans son ensemble, plutôt qu'aux caractères particuliers qui composent la suite.

D9 *Propriété de directionalité :* une propriété de chaque caractère graphique qui précise son ordonnancement horizontal tel que décrit la section *3.12, Comportement bidirectionnel.*

- La composition des écritures droite-à-gauche, comme l'arabe ou l'hébreu, nécessite l'interprétation des propriétés directionnelles conformément à l'algorithme bidirectionnel Unicode.

D10 *Propriété miroir :* propriété des caractères dont l'image est réfléchie horizontalement dans les textes disposés de droite à gauche (par rapport à leur image de gauche à droite). Voir la section *4.7, Caractères miroirs – normatif*.

- En d'autres mots, on interprète U+0028 (PARENTHÈSE GAUCHE comme une parenthèse ouvrante ; dans un contexte gauche-à-droite, ce caractère s'affiche sous la forme d'une « (» ; par contre dans un contexte de droite-à-gauche, ce caractère serait réfléchi et apparaîtrait comme une «) ».

- Il s'agit du comportement implicite adopté dans les textes Unicode (pour plus d'informations, reportez-vous à la section *7.1, Ponctuation générale*).

D10a *Propriété de casse :* une propriété des caractères de certains alphabets pour lesquels certains caractères sont considérés comme une variante d'une même lettre (cf. la section *4.1, Casse – normatif*).

D10b *Propriété de valeur numérique :* une propriété des caractères numéraux, c'est-à-dire ceux qui désignent des nombres (cf. la section *4.6, Valeur numérique – normatif*).

D11 *Propriétés de caractères spéciaux :* le comportement de la plupart des caractères ne mérite pas une attention particulière de la part d'Unicode. Cependant, certains caractères manifestent un comportement spécial, celui-ci est expliqué dans la description de bloc de caractères correspondant à ces caractères. La section *3.9, Propriétés des caractères spéciaux,* énumère pour sa part ces caractères.

D12 *Usage privé :* Les numéros Unicode de U+E000 à U+F8FF et de U+F0000 à U+10FFFF – ces derniers étant représentés en UTF-16 par des paires de seizets (voir la section *3.7, Seizets d'indirection*) dont le seizet d'indirection supérieur appartient à l'intervalle de DB80 à DBFF – sont réservés à l'usage privé.

3.5 Combinaison

D13 *Caractère de base :* un caractère qui ne se combine pas graphiquement avec des caractères qui le précèdent et qui n'est ni un caractère de commande ni de formatage.

- La plupart des caractères sont des caractères de base. Cette absence de combinaison graphique de la part des caractères de base ne leur interdit pas d'adopter des formes contextuelles ou de prendre part à des ligatures.

D14 *Caractère combinatoire :* un caractère qui se combine graphiquement avec un caractère de base qui le précède. On dit que le caractère combinatoire est adjoint au caractère de base.

- Ces caractères ne s'utilisent pas de manière isolée (à moins qu'on veuille les décrire). Ils comprennent les accents, les points hébreux, les points-voyelles arabes, les matras indiennes et autres diacritiques.

- Bien qu'un caractère combinatoire soit destiné à s'adjoindre graphiquement à un caractère de base, il existe des circonstances où soit (1) aucun caractère de base ne précède le caractère combinatoire, soit (2) un processus se révèle incapable d'effectuer la combinaison graphique. Dans les deux cas, le processus peut afficher le caractère combinatoire sans combinaison graphique ; en d'autres mots, il peut le représenter comme s'il s'agissait d'un caractère de base.

- Les images représentatives des caractères combinatoires sont illustrées à l'aide d'un cercle en pointillés dans les tableaux de code ; quand un caractère combinatoire doit s'adjoindre à un caractère de base précédent, sa position relative au cercle pointillé indique la position qu'il devrait prendre par rapport à ce caractère de base.

- Les caractères combinatoires héritent habituellement les propriétés de leur caractère de base, tout en conservant leurs propriétés combinatoires.

- Les caractères de commande et de formatage, tels que les *tabulations* ou les *signes de gauche-à-droite* ne sont pas des caractères de base. Les caractères combinatoires ne s'y adjoignent pas.

D15 *Signe à chasse nulle :* un caractère combinatoire dont le positionnement lors de la présentation dépend de son caractère de base. Habituellement, ce caractère n'occupe pas de place par lui-même le long de la ligne de base visuelle.

- Ces caractères peuvent toutefois être suffisamment larges pour affecter la position de leur caractère de base par rapport aux caractères de base voisins. Ainsi, un circonflexe adjoint à un « i » pourra-t-il augmenter la chasse de « î », il en va de même pour le caractère U+20DD ◌⃝ DIACRITIQUE CERCLE ENGLOBANT.

D16 *Signe avec chasse :* un caractère combinatoire qui n'est pas un signe à chasse nulle.

- Exemple : 093F ि VOYELLE DIACRITIQUE DÉVANÂGARÎ I. En général, le comportement des signes avec chasse ne diffère guère de celui des caractères de base.

D17 *Suite de caractères combinatoires :* une suite de caractères composée soit d'un caractère de base suivi d'une suite d'un ou plusieurs caractères combinatoires, soit d'une suite d'un ou plusieurs caractères combinatoires.

- Une suite de caractères combinatoires se nomme également une suite de caractères composites.

D17a *Suite défectueuse de caractères combinatoires :* une suite de caractères combinatoires qui ne commence pas par un caractère de base.

- Une suite défectueuse de caractères combinatoires se présente lorsqu'une suite de caractères combinatoires apparaît au début d'une chaîne ou suit un caractère de formatage ou de commande.

3.6 Décomposition

D18 *Caractère décomposable :* caractère équivalent à une suite d'un ou plusieurs autres caractères selon les décompositions indiquées dans la liste de noms de la section *15.1, Liste de noms de caractère*. On le nomme également *caractère précomposé* ou caractère *composite.*

D19 *Décomposition :* suite d'un ou plusieurs caractères équivalente à un caractère décomposable. La décomposition complète d'une suite de caractères résulte de la décomposition de chacun des caractères de la suite jusqu'à ce qu'aucun caractère ne puisse plus être décomposé.

Décomposition de compatibilité

D20 *Décomposition de compatibilité complète*[14] *:* décomposition d'un caractère qui découle de l'application récursive à la fois des décompositions canoniques et de compatibilité précisées dans la liste des noms de la section *15,1 Liste de noms de caractère* et de celles décrites à la section *3.11, Comportement des jamos jointifs*, jusqu'à ce qu'aucun caractère ne puisse plus être décomposé, et enfin du réordonnancement des signes à chasse nulle conformément à la section *3.10, Mise en ordre canonique*.

- Il est possible qu'une décomposition de compatibilité amène la perte d'informations de formatage.

[14] Décomposition complète qu'il faut distinguer des décompositions simples de compatibilité telle qu'on les trouve dans la *Base de données de caractères Unicode* (qui ne sont pas complètes).

D21 *Caractère de compatibilité :* caractère auquel est associé au moins une décomposition de compatibilité.

- Le standard Unicode prévoit des décompositions de compatibilité afin de préserver des distinctions présentes dans d'autres normes. Ces caractères permettent donc la transmission et le traitement de données préexistantes.

- Le remplacement d'un caractère de compatibilité par sa décomposition peut entraîner la perte de la convertibilité aller-retour avec une norme de base.

D22 *Équivalent de compatibilité* : on dit que deux suites de caractères sont équivalentes de compatibilité si leurs décompositions de compatibilité complètes sont identiques.

Décomposition canonique

D23 *Décomposition canonique complète :* décomposition d'un caractère qui découle de l'application récursive des décompositions canoniques précisées dans la liste des noms de la section *15.1, Liste des noms de caractères* et de celles décrites à la section *3.11, Comportement des jamos jointifs*, jusqu'à ce qu'aucun caractère ne puisse plus être décomposé, et enfin du réordonnancement des signes à chasse nulle conformément à la section *3.10, Mise en ordre canonique*.

- Les décompositions canoniques constituent un sous-ensemble des décompositions de compatibilité ; toutefois une décomposition canonique n'élimine pas d'information de formatage.

D24 *Équivalent canonique :* on dit de deux suites de caractères qu'elles sont canoniquement équivalentes si leurs décompositions canoniques complètes sont identiques.

- Ainsi, les suites <*o, tréma diacritique*> et <*ö*> sont-elles canoniquement équivalentes. L'équivalence canonique est une propriété Unicode. Elle ne doit pas être confondue avec des équivalences de tri ou de correspondance propres aux langues qui constituent des équivalences supplémentaires. En suédois, par exemple, le *ö* est une lettre totalement différente du *o* qui se classe après *z.* En allemand, le *ö* est faiblement équivalent à *oe* et se classe avec le *oe*. En anglais, *ö* n'est qu'un *o* coiffé d'un diacritique indiquant qu'il se prononce séparément de la lettre précédente (comme dans *coöperate)*, il se classe avec *o*.

Remarque : pour des définitions de composition canonique et composition de compatibilité, veuillez consulter le chapitre *6, Formes de normalisation Unicode*.

3.7 Seizets d'indirection

D25 *Seizet d'indirection supérieur :* en UTF-16, unité de stockage d'Unicode comprise dans l'intervalle de D800 à DBFF.

D26 *Seizet d'indirection inférieur :* en UTF-16, unité de stockage d'Unicode comprise dans l'intervalle de DC00 à DFFF.

D27 *Paire de seizets d'indirection :* représentation codée d'un caractère abstrait qui consiste en une paire d'unités de stockage dont la première valeur est un seizet d'indirection supérieur et la seconde est un seizet d'indirection inférieur.

- Contrairement aux caractères combinatoires, qui possèdent une sémantique et des propriétés indépendantes, les seizets d'indirection n'ont pas de sens particulier quand ils ne forment pas une paire d'indirection.

- Les paires de seizets d'indirection permettent de représenter en UTF-16 des caractères qui feront partie des prochaines extensions du standard Unicode. Il n'existe pas actuellement de caractères affectés accessibles par ces seizets, mais tout laisse penser que d'ici peu de nouveaux caractères seront ajoutés aux plans supplémentaires. Pour plus de renseignements, voir les sections *14.4, Zone d'indirection* et *5.4, Traitement des paires d'indirection*.

D28 *Valeur scalaire Unicode :* nombre N de 0 à $10FFFF_{16}$ défini par l'application de l'algorithme suivant à une suite de caractère V [15]:

N = U	si V est une valeur simple et pas un seizet d'indirection <U>
N = $(S - D800_{16}) * 400_{16} + (I - DC00_{16}) + 10000_{16}$	si V est une paire de seizets d'indirection <S,I>

- Le standard Unicode définit des valeurs scalaires Unicode afin de pouvoir utiliser Unicode dans des normes comme SGML, XML et HTML qui exigent que les caractères abstraits aient une valeur scalaire (en d'autres mots, sans indirection).

- Cet algorithme est identique à celui de l'ISO/CEI 10646 utilisé pour transformer l'UTF-16 en UCS-4 (pour plus de renseignements, voir l'annexe *C, Comparaison entre ISO/CEI 10646 et Unicode*).

- La transformation inverse définit la paire de seizets d'indirection à partir d'une valeur scalaire Unicode de la façon suivante :

  $S = (V - 10000_{16}) / 400_{16} + D800_{16}$
  $I = (V - 10000_{16}) \% 400_{16} + DC00_{16}$

  Les opérateurs « / » et « % » sont définis dans la section *0.2, Conventions de notation*.

- Une valeur scalaire Unicode porte souvent le nom de position de code ou de point de code dans l'industrie informatique[16].

3.8 Transformations

Plus d'une représentation de données Unicode peut être conforme au standard Unicode. Au premier chef, on trouve UTF-8 (voir section *2.3, Modèles de caractères* et Annexe *C.3, Formats de transformation JUC*). Il existe de surcroît des surcodages de transfert (généralement des codages de compression) comme ceux décrits dans le rapport technique Unicode n°6 « A Standard Compression Scheme for Unicode » repris sur le disque qui accompagne ce livre ou sur le site Internet du consortium Unicode pour la dernière version.

D29 Un *format transformé d'Unicode (ou du JUC), abrégé en UTF*, transforme chaque numéro de caractère Unicode en une suite unique d'unités de stockage. Un UTF peut

[15] La manière d'énoncer cette notion, comme dérivant des seizets d'indirection, est d'origine historique. En fait, il est plus simple et plus logique de considérer la valeur scalaire Unicode comme le numéro affecté à un caractère abstrait et de définir les valeurs de seizets à partir de ces numéros, plutôt que l'inverse.

[16] Le terme le plus simple est sans doute *numéro de code* et, quand un caractère y est affecté, *numéro de caractère*.

spécifier l'ordre des octets à adopter lors de la sérialisation unités de stockage. Un UTF peut également préciser l'utilisation d'un *indicateur d'ordre des octets*.

- Le format transformé précise le type d'unités de stockage — il peut s'agir, par exemple, de seizets ou d'octets.

- Toute suite d'unités de stockage qui pourrait correspondre à numéro plus grand que $10FFFF_{16}$ est illégale.

Puisque chaque suite de caractères codés Unicode correspond à une seule suite d'unités de stockage pour un UTF donné, il est donc possible de calculer la correspondance inverse. Ainsi chaque UTF permet-il un transcodage aller-retour sans perte de n'importe quelle suite de caractères codés Unicode C vers une suite d'unités de stockage S ; on peut à partir de S recalculer C. Afin de garantir l'existence de ce transcodage aller-retour, une transformation UTF (sauf UTF-16[18]) doit donc également transformer les numéros erronés (par ex. $xxFFFE_{16}$, $xxFFFF_{16}$ et les seizets d'indirection non appariés) vers des suites d'unités de stockage uniques.

D30 Pour un UTF donné, une suite d'unités de stockage qui ne peut être produite à partir d'une suite de numéros Unicode est appelée une *suite mal formée d'unités de stockage*.

D31 Pour un UTF donné, une suite d'unités de stockage qui ne correspond à aucune suite de numéros Unicode est appelée une *suite illégale d'unités de stockage*.

- Ainsi, en UTF-8 chaque unité de stockage de la forme $110xxxxx_2$ *doit* être suivie d'une unité de stockage de la forme $10xxxxxx_2$. Une suite du type $110xxxxx_2$ $0xxxxxxx_2$ est illégale et ne peut jamais être générée. Confronté à cette suite illégale d'unités de stockage, un processus UTF-8 conforme peut traiter la première unité de stockage $110xxxxx_2$ comme une erreur de terminaison illégale et, par exemple, signaler une erreur, éliminer l'unité de stockage ou représenter celle-ci par un repère comme U+FFFD ◆ CARACTÈRE DE REMPLACEMENT. Dans les deux derniers cas, ce processus doit continuer le traitement à partir de la deuxième unité de stockage $0xxxxxxx_2$.

D32 Pour un UTF donné, toute suite mal formée d'unités de stockage qui n'est pas illégale est appelée une suite irrégulière d'unités de stockage.

- Afin de faciliter et d'accélérer les mises en œuvre, certains formats transformés peuvent permettre la présence des suites irrégulières d'unités de stockage sans qu'un traitement d'erreur ne soit nécessaire. Un processus conforme doit s'abstenir d'utiliser des suites irrégulières d'unités de stockage pour coder des informations hors bande.

D33 UTF-16BE est un format transformé d'Unicode qui sérialise une valeur Unicode sous la forme d'une suite de seizets en format grand-boutien. Une suite initiale qui correspond à U+FEFF est interprétée comme une ESPACE INSÉCABLE SANS CHASSE.

- En UTF-16BE, <004D 0061 0072 006B> se sérialise en <00 4D 00 61 00 72 00 6B>.

D34 UTF-16LE est un format transformé d'Unicode qui sérialise une valeur Unicode sous la forme d'une suite de seizets en format petit-boutien. Une suite initiale qui correspond à U+FEFF est interprétée comme une ESPACE INSÉCABLE SANS CHASSE.

- En UTF-16LE, <004D 0061 0072 006B> se sérialise en <4D 00 61 00 72 00 6B 00>.

[18] Car la suite de valeurs scalaires U+DC00 U+D800 U+DC00 U+D800 sera codée en UTF-16 de la même manière et pourra être décodée comme étant équivalente aux valeurs scalaires suivantes U+DC00 U+10000 U+D800.

D35 UTF-16 est un format transformé d'Unicode qui sérialise les valeurs Unicode en une suite de seizets, en format petit-boutien ou grand-boutien. Une suite initiale d'octets correspondant à U+FEFF s'interprète comme un *indicateur d'ordre des octets* : il permet de départager entre les deux formats de sérialisation. On ne considère pas l'*indicateur d'ordre des octets* comme faisant partie du contenu textuel. Une sérialisation de valeurs Unicode selon UTF-16 peut, de manière facultative, commencer par un *indicateur d'ordre des octets*.

- En UTF-16, <004D 0061 0072 006B> se sérialise en <FF FE 4D 00 61 00 72 00 6B 00>, <FE FF 00 4D 00 61 00 72 00 6B> ou encore <00 4D 00 61 00 72 00 6B>.

- Le terme *UTF-16* peut être ambigu. Quand on parle du codage d'Unicode en mémoire (la forme de stockage), on n'associe pas d'orientation particulière aux octets et on n'utilise jamais l'IOO. Alors que, lorsqu'on parle de la sérialisation des valeurs Unicode en octets, un IOO peut être présent et adopter les deux orientations d'octet.

D36 (a) UTF-8 est un format transformé Unicode qui sérialise une valeur scalaire Unicode en une suite d'un à quatre octets, conformément au *Tableau 3-1, Distribution binaire des octets en UTF-8*.

(b) On nomme *suite d'unités de stockage UTF-8 illégale* toute suite d'octets qui ne correspond à aucun des modèles énumérés au *Tableau 3-1B, Suites valables d'octets UTF-8*.

(c) Les suites d'unités de stockage irrégulières UTF-8 sont composées de six octets dont les trois premiers correspondent au seizet d'indirection supérieur et les trois derniers au seizet d'indirection inférieur. Conformément à la clause C12, aucun processus conforme ne doit générer de suites UTF-8 irrégulières.

- En UTF-8, <004D 0061 0072 006B> se sérialise en <4D 61 72 6B>.

- Les problématiques suites d'octets UTF-8 « non minimales » correspondaient aux caractères PMB à représentation multiple. Une seule représentation est permise, les autres suites sont illégales car elles ne sont pas permises par le *Tableau 3.1B*.

Le *tableau 3-1* précise la distribution des bits pour différents numéros Unicode en des suites UTF-8 composées d'un à quatre octets. Remarquez que la forme à quatre octets pour les paires d'indirection implique l'ajout de 10000_{16} pour prendre en compte le décalage initial des valeurs codées auxquelles font référence ces paires d'indirection[19]. Pour une discussion sur la formulation différente du format UTF-8 pour l'ISO/CEI 10646, veuillez vous référer à la section *C.3, Formes transformées du JUC*.

[19] Une nouvelle fois, il nous paraît plus simple de considérer le calcul à partir du numéro de caractère (la « valeur scalaire Unicode ») plutôt que d'un format transformé particulier comme l'UTF-16. La transformation en UTF-8 en est alors simplifiée.

Tableau 3-1. Distribution binaire des octets en UTF-8

Valeur scalaire	UTF-16	1er octet	2e octet	3e octet	4e octet
00000000 0xxxxxxx	00000000 0xxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
000uuuuu zzzzyyyy yyxxxxxx	110110ww wwzzzzyy[a] + 110111yy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

a. Où wwww = uuuuu - 1 (pour prendre en compte l'ajout de 10000_{16}, voir la section *3.7, Seizets d'indirection*).

Tableau 3-1B. Suites valables d'octets UTF-8

Points de code	1er octet	2e octet	3e octet	4e octet
U+0000..U+007F	00..7F			
U+0800..U+07FF	C2..DF	80..BF		
U+0800..U+0FFF	E0	A0..BF	80..BF	
U+1000..U+FFFF	E1..EF	80..BF	80..BF	
U+10000..U+3FFFF	F0	90..BF	80..BF	80..BF
U+40000..U+FFFFF	F1..F3	80..BF	80..BF	80..BF
U+100000..U+10FFFF	F4	80..8F	80..BF	80..BF

Le *Tableau 3-1B* énumère toutes les suites d'octets valables en UTF-8. Un intervalle de valeurs d'octet tels que A0..BF signifie que tout octet de A0 à BF (y compris ces deux valeurs) est valable à la position mentionnée. La suite d'octets <C0,AF>, par exemple, est *illégale* car C0 n'est pas permis comme premier octet. La suite d'octets <E0, 9F, 80> est-elle aussi *illégale* car, dans la rangée où le E0 est valable comme premier octet, 9F n'est, pour sa part, pas admis comme deuxième octet. Par contre, la suite d'octets <F4, 80, 83, 92> est valable car chaque octet de la suite appartient à un intervalle d'octets dans une même rangée du tableau (la dernière rangée).

- Les cas où l'intervalle d'un octet postérieur ne correspond pas à 80..BF sont soulignés. Seuls les deuxièmes octets d'une suite présentent parfois ces intervalles de validité irréguliers.

3.9 Propriétés des caractères spéciaux

Le comportement de la plupart des caractères ne requiert pas un examen particulier de la part de ce standard. Cependant, les caractères suivants manifestent un comportement particulier (voir les chapitres *14, Zones spéciales et caractères de formatage* et *4, Propriétés des caractères*).

- Commandes de fin de ligne

  0009 TABULATION HORIZONTALE
  000A PASSAGE À LA LIGNE
  000C SAUT DE PAGE

000D RETOUR DE CHARIOT
0020 ESPACE
00A0 ESPACE INSÉCABLE
0F0B SIGNE TIBÉTAIN INTERSYLLABIQUE TSHEG
0F0C SIGNE DÉLIMITEUR TIBÉTAIN TSHEG BSTAR
2000 DEMI-CADRATIN
2001 CADRATIN
2002 ESPACE DEMI-CADRATIN
2003 ESPACE CADRATIN
2004 TIERS DE CADRATIN
2005 QUART DE CADRATIN
2006 SIXIÈME DE CADRATIN
2007 ESPACE NUMÉRIQUE
2008 ESPACE PONCTUATION
2009 ESPACE FINE
200A ESPACE ULTRA-FINE
200B ESPACE SANS CHASSE
2011 TRAIT D'UNION INSÉCABLE
2028 SÉPARATEUR DE LIGNES
2029 SÉPARATEUR DE PARAGRAPHES
202F ESPACE INSÉCABLE ÉTROITE
FEFF ESPACE INSÉCABLE SANS CHASSE

- Commande de coupe de mots
  - 002D TRAIT-D'UNION-SIGNE-MOINS
  - 00AD TRAIT D'UNION VIRTUEL
  - 058A TRAIT D'UNION ARMÉNIEN
  - 1806 TRAIT D'UNION VIRTUEL MONGOL TODO
  - 2010 TRAIT D'UNION
  - 2011 TRAIT D'UNION INSÉCABLE
  - 2027 POINT DE COUPURE DE MOT

- Formatage de fraction
  - 2044 BARRE DE FRACTION

- Comportement spécial avec des signes à chasse nulle
  - 0020 ESPACE
  - 0069 LETTRE MINUSCULE LATINE I
  - 006A LETTRE MINUSCULE LATINE J
  - 00A0 ESPACE INSÉCABLE
  - 0131 LETTRE MINUSCULE LATINE I SANS POINT

- Signes doubles à chasse nulle
  - 0360 DIACRITIQUE DOUBLE TILDE
  - 0361 DIACRITIQUE DOUBLE BRÈVE RENVERSÉE
  - 0362 DIACRITIQUE DOUBLE FLÈCHE VERS LA DROITE SOUSCRITE

- Liants
  - 200C ANTI-LIANT SANS CHASSE
  - 200D LIANT SANS CHASSE

- Ordre bidirectionnel
  - 200E MARQUE GAUCHE-À-DROITE
  - 200F MARQUE DROITE-À-GAUCHE
  - 202A ENCHÂSSEMENT GAUCHE-À-DROITE

202B ENCHÂSSEMENT DROITE-À-GAUCHE
202C DÉPILEMENT DE FORMATAGE DIRECTIONNEL
202D FORÇAGE GAUCHE-À-DROITE
202E FORÇAGE DROITE-À-GAUCHE

- Basculement de formatage
  206A INHIBITEUR D'ÉCHANGE SYMÉTRIQUE
  206B ACTIVATEUR D'ÉCHANGE SYMÉTRIQUE
  206C INHIBITEUR DE FORMAGE ARABE
  206D ACTIVATEUR DE FORMAGE ARABE
  206E SÉLECTEUR DE FORMES DE CHIFFRE NATIONALES
  206F SÉLECTEUR DE FORMES DE CHIFFRE NOMINALES

- Abréviation syriaque
  070F SYMBOLE D'ABRÉVIATION SYRIAQUE

- Formation de consonne morte dans les écritures brâhmî
  094D SYMBOLE DÉVANÂGARÎ VIRÂMA
  09CD SYMBOLE BENGALI VIRÂMA
  0A4D SYMBOLE GOURMOUKHÎ VIRÂMA
  0ACD SYMBOLE GOUDJARATI VIRÂMA
  0B4D SYMBOLE ORIYA VIRÂMA
  0BCD SYMBOLE TAMOUL VIRÂMA
  0C4D SYMBOLE TÉLOUGOU VIRÂMA
  0CCD SYMBOLE KANNARA VIRÂMA
  0D4D SYMBOLE MALAYALAM VIRÂMA
  0DCA SYMBOLE SINGHALAIS AL-LAKOUNA
  0E3A LETTRE THAÏE PHINTHOU
  0F84 SIGNE TIBÉTAIN HALANTA
  1039 SYMBOLE BIRMAN VIRÂMA
  17D2 SIGNE KHMER TCHOENG

- Sélecteur de variantes mongoles
  180B SÉLECTEUR DE VARIATION LIBRE MONGOL UN
  180C SÉLECTEUR DE VARIATION LIBRE MONGOL DEUX
  180D SÉLECTEUR DE VARIATION LIBRE MONGOL TROIS
  180E SÉPARATEUR DE VOYELLES MONGOL

- Indicateur de variation idéographique
  303E INDICATEUR DE VARIATION IDÉOGRAPHIQUE

- Description idéographique
  2FF0 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE GAUCHE À DROITE
  2FF1 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE DE HAUT EN BAS
  2FF2 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE DE LA GAUCHE AU MILIEU PUIS À LA DROITE
  2FF3 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE DU HAUT AU MILIEU PUIS EN BAS
  2FF4 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE TOUT AUTOUR
  2FF5 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE AUTOUR À PARTIR DU HAUT
  2FF6 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE AUTOUR À PARTIR DU BAS
  2FF7 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE AUTOUR À PARTIR DE LA GAUCHE
  2FF8 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE AUTOUR À PARTIR D'EN HAUT À GAUCHE
  2FF9 CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE AUTOUR À PARTIR D'EN HAUT À DROITE

2FFA CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE AUTOUR À PARTIR D'EN BAS À GAUCHE
2FFB CARACTÈRE DE DESCRIPTION IDÉOGRAPHIQUE CHEVAUCHEMENT

- Annotation interlinéaire
  FFF9 ANCRE D'ANNOTATION INTERLINÉAIRE
  FFFA SÉPARATEUR D'ANNOTATION INTERLINÉAIRE
  FFFB TERMINATEUR D'ANNOTATION INTERLINÉAIRE

- Remplacement d'objet
  FFFC CARACTÈRE DE REMPLACEMENT D'OBJET

- Repli de conversion de code
  FFFD CARACTÈRE DE REMPLACEMENT

- Signature d'ordre d'octet
  FEFF ESPACE INSÉCABLE SANS CHASSE

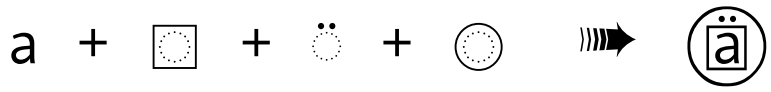
3.10 Mise en ordre canonique

L'objectif de cette section est de préciser une manière univoque d'interpréter une suite de caractères combinatoires. Dans le standard Unicode, on interprète l'ordre des caractères d'une suite de caractères combinatoires selon les principes suivants :

- Tous les caractères combinatoires d'un texte Unicode suivent le caractère de base auquel ils s'adjoignent. Ainsi, c'est de manière non ambigüe qu'on interprète (et affiche) la suite Unicode U+0061 a LETTRE MINUSCULE LATINE A + U+0308 ◌̈ DIACRITIQUE TRÉMA + 0075 u LETTRE MINUSCULE LATINE U comme « äu » et non « aü ».

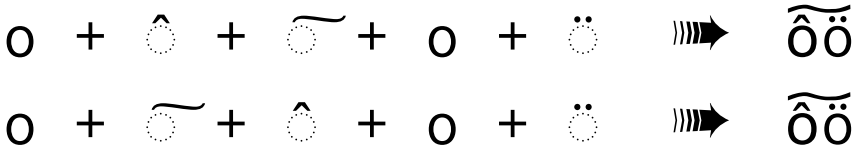
- Les signes sans chasse englobants encadrent tous les caractères précédents, y compris le caractère de base (voir *Figure 3-1*). Ils encadrent ainsi successivement les signes sans chasse englobants antérieurs.

Figure 3-1. Signes englobants



- Les diacritiques doubles s'adjoignent toujours de manière plus lâche que les autres signes sans chasse. Au rendu, le diacritique double doit coiffer les autres diacritiques, à l'exception des diacritiques englobants (voir Figure 3.-2)

Figure 3-2. Positionnement des doubles diacritiques



- Les signes combinatoires de même classe combinatoire s'éloignent généralement graphiquement du caractère de base qu'ils modifient. Certains signes combinatoires particuliers dérogent à l'empilement implicite et se placent côte à côte plutôt que l'un sur l'autre ou forment une ligature avec le signe sans chasse adjacent. Quand ils sont

placés côte à côte, l'ordre dans lequel apparaissaient les caractères codés est respecté dans le positionnement de leurs glyphes relatifs et cela dans l'ordre dominant de l'écriture utilisée.

- Si les caractères combinatoires sont de classes combinatoires différentes — c'est le cas lorsqu'un signe combinatoire coiffe un caractère de base alors qu'un autre se place sous le caractère de base — l'ordre des deux diacritiques en question n'entraîne aucune distinction graphique ni sémantique.

Afin d'éviter toute ambigüité dans l'interprétation des suites combinatoires, les sous-sections suivantes précisent de manière formelle ces principes à l'aide d'une liste normative de classes combinatoires et d'une déclaration algorithmique qui précise l'emploi de ces classes combinatoires.

Classes combinatoires

Le standard Unicode considère les suites de signes sans chasse comme équivalentes si ces signes n'interagissent pas au niveau typographique. L'algorithme de mise en ordre canonique définit une méthode pour établir quelles suites interagissent et fournit un ordre canonique pour ces séquences à utiliser pour établir l'équivalence.

D37 *Classe combinatoire* : valeur numérique attribuée à chaque caractère combinatoire Unicode ; cette valeur détermine l'ensemble des caractères combinatoires avec lequel ce premier caractère combinatoire interagit typographiquement.

- Voir la section *4.2, Classes combinatoires – normatif*, pour une liste complète des classes combinatoires des caractères Unicode.

On dit que deux caractères sont de même classe si et seulement s'ils interagissent au niveau typographique.

- Les caractères englobants ainsi que les caractères combinatoires avec chasse appartiennent à la classe combinatoire de leur caractère de base.

- La valeur numérique associée à une classe combinatoire n'a pas de signification particulière ; l'unique dessein dans l'attribution de ces valeurs est de distinguer les classes combinatoires pour faciliter les comparaisons d'équivalence de suites de caractères combinatoires.

Ordre canonique

L'ordre canonique d'une suite décomposée de caractères découle d'un processus de tri effectué sur chaque suite de caractères combinatoires d'après leurs classes combinatoires. On trie les caractères appartenant à la classe combinatoire zéro indépendamment des autres caractères, la somme de travail à accomplir par l'algorithme dépend donc de la longueur des suites de caractères de classes non zéro. Pour un texte typique, l'exécution de cet algorithme sera extrêmement rapide.

L'algorithme présenté ci-dessous décrit la logique du processus de mise en ordre. Cette description n'exclut pas des mises en oeuvre optimisées de cet algorithme pour autant qu'elles lui soient équivalentes — c'est-à-dire qu'elles produisent le même résultat.
Plus exactement, l'ordre canonique d'une suite décomposée de caractères D est défini par l'algorithme suivant :

R1 *Pour chaque caractère x dans D, soit p(x) la classe combinatoire de x.*

R2 Pour chaque paire (a,b) de caractères voisins dans D qui vérifie la condition p(b) ≠ 0 & p(a) > p(b), intervertir ces caractères.

R3 Répéter l'opération R2 jusqu'à ce que l'on ne puisse plus intervertir de caractères de D.

Le *tableau 3-2* donne quelques exemples d'ordonnancement canonique.

Tableau 3-2. Quelques classes combinatoires

Classe combinatoire	Abréviation	Code	Nom ISO 10646
0	a	0061	LETTRE MINUSCULE LATINE A
220	point-souscrit	0323	DIACRITIQUE POINT SOUSCRIT
230	tréma	0308	DIACRITIQUE TRÉMA
230	brève	0306	DIACRITIQUE BRÈVE
0	a-point-souscrit	1EA1	LETTRE MINUSCULE LATINE A POINT SOUSCRIT
0	a-tréma	00E4	LETTRE MINUSCULE LATINE A TRÉMA
0	a-brève	0103	LETTRE MINUSCULE LATINE A BRÈVE

a + point-souscrit + tréma ≡ a + point-souscrit + tréma

a + tréma + point-souscrit ≡ a + point-souscrit + tréma

Point-souscrit appartient à une classe combinatoire inférieure à celle du *tréma*, l'algorithme renvoie donc *a* suivi du *point-souscrit* et du *tréma*. Dans l'exemple ci-dessous, par contre, le *tréma* et la *brève* ne changent pas de place, car ils appartiennent à la même classe combinatoire (ils interagissent donc au niveau typographique).

a + brève + tréma ≢ a + tréma + brève

a + tréma + brève ≢ a + brève + tréma

L'application de l'algorithme donne les résultats illustrés au *Tableau 3-3*.

Tableau 3-3 Mises en ordre canonique

Origine	Décomposé	Trié	Résultat
a-tréma + point-souscrit	a + tréma + point-souscrit	a + point-souscrit + tréma	a + point-souscrit + tréma
a + tréma + point souscrit		a + point-souscrit + tréma	a + point-souscrit + tréma
a + point-souscrit + tréma			a + point-souscrit + tréma
a-point-souscrit + tréma	a + point-souscrit + tréma		a + point-souscrit + tréma
a-tréma + brève	a + tréma + brève		a + tréma + brève
a + tréma + brève			a + tréma + brève
a + brève + tréma			a + brève + tréma
a-brève + tréma	a + brève + tréma		a + brève + tréma

Ordonnancement et tri lexicographique

Lorsqu'un tri culturellement correct ne s'impose pas hors d'un domaine particulier, le processus de tri lexicographique peut omettre, pour les caractères exclus, l'invocation à l'algorithme de mise en ordre canonique. Ainsi, un processus de tri lexicographique grec peut ne pas avoir besoin de trier correctement les lettres cyrilliques ; dans ce cas, il n'est pas

nécessaire qu'il effectue une décomposition maximale ni qu'il réordonne les lettres cyrilliques, il peut se contenter de les trier selon l'ordre Unicode. Pour un traitement plus complet du tri lexicographique, veuillez vous référer au rapport technique Unicode n° 10, « Unicode Collation Algorithm » sur le disque ou sur le site Internet Unicode pour une version tenue à jour.

Il existe également une norme ISO/CEI 14651, intitulée *Classement international de chaînes de caractères — Méthode de comparaison de chaînes de caractères et description du modèle commun et adaptable de classement*.

L'UTR-10 est un « profil » de la norme ISO/CEI 14651. Cette dernière a comme caractéristique principale de ne pas imposer de table implicite, pour qu'un processus soit conforme il faut explicitement déclarer un sous-tableau (un « delta ») pour la langue et pour la culture cibles. En cela, ISO/CEI 14651 reconnaît la nécessité d'adapter l'algorithme implicite à la culture et à la langue locales malgré une table qui propose au départ un modèle valable pour la majorité des langues. Selon cette optique donc aucun tri n'est valable pour toutes les langues.

L'UTR-10 propose une table implicite utilisée en l'absence d'autres indications et il n'insiste pas sur la notion d'adaptabilité, bien que certaines mises en œuvre intelligentes s'y astreindront par le jeu de la concurrence et de l'expression des besoins des acheteurs perspicaces. Il existe une multitude de différences entre l'UTR-10 et l'ISO/CEI 14561 en ce qui a trait aux caractères spéciaux, mais en ce qui concerne le classement alphabétique le résultat est essentiellement le même que celui obtenu avec la table de base de la 14651 en déclarant toutefois un delta minimal.

3.11 Comportement des jamos jointifs

Le standard Unicode comprend à la fois un jeu complet de syllabes hangûl modernes précomposées et un jeu plus restreint de jamos hangûl jointifs qui peut être employé pour coder des blocs syllabiques archaïques ainsi que les blocs syllabiques modernes. Cette section précise comment :

- Trouver les frontières syllabiques dans une suite de caractères jamos jointifs ;
- Composer des syllabes hangûl à partir de caractères jamos ;
- Effectuer la décomposition canonique d'une syllabe hangûl ;
- Établir algorithmiquement les noms correspondant aux caractères syllabiques hangûl.

(Pour plus de renseignements, voir les sous-sections « Syllabes hangûl » et « Jamos hangûl » dans la section *10.4, Hangûl.*)

Les caractères jamos peuvent être classés en trois séries de caractères : *tch'ôsong* (consonne initiale ou caractère syllabique initial), *djoungsong* (voyelle ou caractère syllabique médian) et *djôngsong* (consonne finale ou caractère syllabique final). Par la suite, ces jamos sont représentés par un I (consonne initiale), V (voyelle) et F (consonne finale) ; les limites syllabiques sont représentées par des points médians « · », un X illustre les non-jamos.

Frontières syllabiques

Au rendu, une suite de jamos s'affiche comme une série de blocs syllabiques. Les règles suivantes précisent comment diviser une suite arbitraire de jamos (y compris des suites non normalisées) en blocs syllabiques. Pour ces règles, on considère la *bourre tch'ôsong* (I_b) comme un caractère *tch'ôsong* et la *bourre djoungsong* (V_b) comme un caractère *djoungsong.*

Au sein d'une suite de caractères, les frontières syllabiques se présentent entre les paires de caractères indiqués au tableau 3-4. Toutes les autres suites de jamos hangûl font partie de la même syllabe. Il faut souligner que, à l'instar des caractères non-jamos, tout signe combinatoire entre deux jamos jointifs empêche ces jamos de former une syllabe.

Tableau 3-4. Règles de coupure de syllabe hangûl

Condition	Exemples
Tout jamo jointif et tout non-jamo	I•X, V•X, F•X, X•I, X•V, X•F
Un *djôngsong* (final) et un *tch'ôsong* (initial)	F•I
Un *djoungsong* (voyelle) et un *tch'ôsong* (initial)	V•I
Un *djôngsong* (final) et un *djoungsong* (voyelle)	F•V

Syllabes régulières

Un bloc syllabique régulier est composé d'une suite de *tch'ôsong* suivi d'une suite de *djoungsong* et d'une suite de *djôngsong* facultative (par ex. S = IV ou IVF). Une suite de syllabes irrégulières peut être transformée en une suite de blocs syllabiques réguliers par l'insertion de bourres *tch'ôsong* et *djoungsong*.

Exemples : Dans le tableau 3-5, la rangée (1) représente les frontières syllabiques pour une suite régulière, la rangée (2) indique les limites syllabiques pour une suite irrégulière, la rangée (3) quant à elle montre comment on peut transformer la séquence (2) en une suite régulière par l'insertion de bourres dans chaque syllabe.

Tableau 3-5. Exemples de frontière syllabique

N°	Séquence		Séquence découpée en syllabes
1	IVFIVIVI$_b$I$_b$VI$_b$V$_b$F	→	IVF•IV•IV•IV$_b$•I$_b$V•I$_b$V$_b$F
2	IIFVIFIFVVII	→	IIF•V•IF•IF•VV•II
3	IIFVIFIFVVII	→	IIV$_b$F•I$_b$V•IV$_b$F•IV$_b$F•I$_b$VV•IIV$_b$

Composition de syllabe hangûl

L'algorithme suivant décrit la manière de composer des syllabes hangûl à partir d'une suite de caractères canoniquement décomposés D. Cette section ne fait qu'aborder la composition et la décomposition, la section *6.16, Hangûl,* examine le sujet en plus de détails. Soulignons le fait que — comme pour les autres caractères non-jamos — tout caractère combinatoire entre deux jamos jointifs empêche la composition de ces jamos.

Définissons d'abord quelques constantes :

```
BaseS     = AC00₁₆
BaseI     = 1100₁₆
BaseV     = 1161₁₆
BaseF     = 11A7₁₆
CompteS   = 11172
CompteI   = 19
```

```
CompteV   = 21
CompteF   = 28
CompteN   = CompteV * CompteF
```

1. Parcourir D en composant tous les jamos jointifs (bloc U+1100) autant que possible, selon les règles de décomposition de compatibilité du bloc U+3130 indiquées au Chapitre *15, Tableaux de codes.* (L'inversion typique de jamos jointifs se produit dans les formes précomposées. Dans ces cas-là, on peut se dispenser de cette étape. Cependant, les données brutes saisies au clavier peuvent être décomposées sous une forme de compatibilité mais non canonique.)

2. Soit i la position courante dans la suite D. Calculer les index suivants qui représentent le nombre ordinal (commençant à zéro) pour chacun des composants syllabiques ainsi que l'index j qui représente la position du dernier caractère d'une syllabe.

```
IndexI  = D[i] - BaseI
IndexV  = D[i+1] - BaseV
IndexF  = D[i+2] – BaseF
j       = i + 2
```

3. Si l'un des deux premiers caractères dépasse les limites (`IndexI < 0` ou `IndexI >= CompteI` ou `IndexV < 0` ou `IndexV >= CompteV`) alors ajouter 1 à i, retourner à l'étape 2 et continuer à partir de là.

4. Si le troisième caractère dépasse les limites (`IndexF <= 0` ou `IndexF >= CompteF`) alors il ne fait pas partie de la syllabe. Réinitialiser les variables suivantes :

```
IndexF = 0
j = i + 1
```

5. Remplacer les caractères de `D[i]` à `D[j]` par la syllabe hangûl S et réinitialiser i à j+1.

```
S = (IndexI * CompteV + IndexV) * CompteF + IndexF + BaseS
```

Exemple. Soient les trois premiers caractères

U+1111 ᄑ HANGÛL TCH'ÔSONG P'IÛP'

U+1171 ᅱ HANGÛL DJOUNGSONG WI

U+11B6 ᆶ HANGÛL DJÔNGSONG RIÛL-HIÛH

Calculer les index suivants,

```
IndexI  = 17
IndexV  = 16
IndexF  = 15
```

et remplacer les trois caractères par

```
S       = [( 17* 21) + 16] * 28 + 15 + BaseS
        = D4DB16
        = 퓔
```

Décomposition de syllabe hangûl

On trouvera ci-dessous la transformation inverse — comment d'une syllabe hangûl S dériver sa décomposition canonique D. La transformation normative de ces caractères équivaut aux décompositions canoniques que l'on retrouve dans les tableaux de code pour d'autres caractères.

1. Calculer l'index de la syllabe
   `IndexS = S – BaseS`

2. Si l'`IndexS` appartient à l'intervalle (`0 <= IndexS < CompteS`), calculer les composants de la manière suivante :
   ```
   I       = BaseI + IndexS / CompteN
   V       = BaseV + (IndexS % CompteN) / CompteF
   F       = BaseF + IndexS % CompteF
   ```
   Les opérateurs « / » et « % » sont définis à la section *0.2, Conventions de notation*.

3. Si `F = BaseF` il n'y a pas de consonne finale remplacer alors S par la suite `<I,V>`. Sinon, il existe une consonne finale, remplacer S par la suite `<I,V,F>`.

Exemple :
```
I       = BaseI + 17
V       = BaseV + 16
F       = BaseF + 15
```
$D4DB_{16} \longrightarrow 1111_{16}\ ,\ 1171_{16}\ ,\ 11B6_{16}$

Noms des syllabes hangûl

On peut dériver les noms de caractères des syllabes hangûl de leur décomposition en commençant le nom par la chaîne `SYLLABE HANGÛL` et en y ajoutant dans l'ordre le nom abrégé de chaque composant de la décomposition (voir la section *4.4, Noms abrégés des jamos – normatif)*. Ainsi, pour U+D4DB, on décompose la syllabe comme dans l'exemple précédent. Ce qui produit la suite suivante de trois caractères :

```
U+1111 HANGÛL TCH'ÔSONG P'IÛP'
U+1171 HANGÛL DJOUNGSONG WI
U+11B6 HANGÛL DJÔNGSONG RIÛL-HIÛH
```

On construit alors le nom du caractère U+D4DB : `SYLLABE HANGÛL P'WILH`. Ce nom de caractère est une propriété normative du caractère.

3.12 Comportement bidirectionnel

Le standard Unicode prescrit un ordre que les caractères doivent respecter en mémoire, on l'appelle *ordre logique*. Quand le texte se présente en lignes horizontales, la plupart des écritures affichent les caractères de gauche à droite. Cependant, pour plusieurs écritures (comme l'arabe et l'hébreu), les textes horizontaux s'écrivent de droite à gauche. Si l'ensemble du texte possède la même direction horizontale, l'ordre d'affichage du texte est sans ambiguïté. Par contre, en présence de texte bidirectionnel (un mélange de texte allant de gauche à droite et de droite à gauche), déterminer l'ordre d'affichage des caractères peut parfois être ambigu.

Cette section décrit l'algorithme utilisé pour établir la directionalité d'un texte Unicode bidirectionnel. Cet algorithme étend le modèle implicite employé actuellement par un certain nombre de mises en œuvre tout en ajoutant une série de codes de formatage explicites réservés à des circonstances particulières. Dans la plupart des cas, il n'est pas nécessaire d'adjoindre au texte des informations supplémentaires pour obtenir l'ordre d'affichage correct.

Il existe des textes où l'ordre bidirectionnel implicite ne suffit pas à produire un texte compréhensible. Afin de résoudre ces cas, Unicode prévoit un petit jeu de codes de formatage directionnel qui permet de préciser l'ordre de rendu des caractères. Ces codes permettent de contrôler de manière précise l'ordre d'affichage et assurent de la sorte un échange lisible. Ils permettent d'ordonner correctement pour l'affichage les textes bruts comme les noms de fichier ou les étiquettes.

Les codes de formatage directionnel ne font qu'*influencer* l'ordre d'affichage du texte. Sous tous les autres rapports, ils devraient être ignorés — ils n'ont aucun effet sur les comparaisons de texte, les coupures de mot, l'analyse lexicale ou numérique.

Quand on traite du texte bidirectionnel, on continue d'interpréter les caractères dans l'ordre logique — seul l'affichage est affecté. L'ordre d'affichage du texte bidirectionnel dépend des propriétés directionnelles des caractères qui composent le texte.

Codes de formatage bidirectionnel

On utilise deux types de codes explicites portant sur des intervalles qui permettent de modifier l'algorithme bidirectionnel implicite d'Unicode : les codes d'enchâssement et de forçage. Il existe, en outre, deux codes d'ordonnancement ponctuels, les marques *de droite-à-gauche* et de *gauche-à-droite*. Tous ces codes ne s'appliquent qu'au paragraphe courant, leur effet prend fin au prochain séparateur de paragraphes. Les types bidirectionnels gauche-à-droite et droite-à-gauche sont qualifiés de *types forts* ; on dit aussi que ces types de caractère sont à forte directionalité. Les types bidirectionnels associés aux chiffres sont qualifiés de *types faibles* ; on dit aussi que ces types de caractère sont à faible directionalité.

Bien qu'on utilise le terme d'*enchâssement* pour certains codes explicites, le texte « enchâssé » par ces paires de codes n'est pas isolé du texte avoisinant. Les caractères enchâssés peuvent modifier l'ordre d'affichage des caractères à l'extérieur de l'enchâssement, et *vice versa*. Cet algorithme est conçu de telle sorte que l'on puisse remplacer les codes explicites de manière équivalente par des informations hors-bande, des feuilles de style par exemple.

Enchâssement directionnel. Ces codes permettent d'indiquer qu'un morceau de texte doit être traité comme s'il était enchâssé et doté d'une certaine directionalité. Ainsi, on pourra baliser un texte français cité au milieu d'une phrase arabe comme du texte gauche-à-droite enchâssé. Si la citation française venait à inclure une phrase en hébreu, cette dernière pourrait alors être marquée comme du texte droite-à-gauche enchâssé. Ces enchâssements peuvent s'imbriquer.

EDAG *(RLE)*	Enchâssement de droite à gauche	Considérer le texte qui suit comme enchâssé et de droite à gauche.
EGAD *(LRE)*	Enchâssement de gauche à droite	Considérer le texte qui suit comme enchâssé et de gauche à droite.

L'examen de l'algorithme clarifiera la signification précise de ces codes. Pour obtenir l'effet d'une ligne écrite de droite à gauche, par exemple, on peut tout simplement ceindre le texte d'un EDAG *(RLE)* et d'un DFD *(PDF)*.

Forçage directionnel prolongé. Les codes suivants permettent de déroger au type de caractère bidirectionnel implicite dans certains cas particuliers comme les numéros de pièce. Il est permis d'imbriquer plusieurs forçages directionnels à l'aide de ces codes.

FDAG *(RLO)*	Forçage droite-à-gauche	Forcer la directionalité des caractères qui suivent : forte directionalité droite-à-gauche.
FGAD *(LRO)*	Forçage gauche-à-droite	Forcer la directionalité des caractères qui suivent : forte directionalité gauche-à-droite.

L'examen de l'algorithme clarifiera la signification précise de ces codes. Le forçage droite-à-gauche, par exemple, permet d'indiquer qu'un numéro de pièce composé de lettres et chiffres latins ainsi que des lettres hébraïques doit s'écrire de droite à gauche.

Fin des codes directionnels prolongés. Le code suivant met fin aux effets du dernier code à effet prolongé explicite (qu'il s'agisse d'un enchâssement ou d'un forçage) et rétablit l'état bidirectionnel antérieur à ce dernier code explicite.

DFD *(PDF)*	Dépilement de formatage directionnel	Rétablir l'état bidirectionnel à ce qu'il était avant le dernier FDAG, FGAD, EDAG ou EGAD.

Marques directionnelles ponctuelles. Ces caractères sont des codes de très peu de poids. Ils agissent exactement comme des caractères gauche-à-droite ou droite-à-gauche, si ce n'est qu'ils ne sont pas affichés et qu'ils n'ont pas d'autre sémantique. Leur utilisation est habituellement plus commode que les enchâssements et les forçages explicites car leur portée est bien plus locale.

MDAG *(RLM)*	Marque droite-à-gauche	Caractère droite-à-gauche de chasse nulle
MGAD *(LRM)*	Marque gauche-à-droite	Caractère gauche-à-droite de chasse nulle

L'algorithme qui suit ne mentionne pas les marques directionnelles ponctuelles, car leur effet sur l'ordre bidirectionnel est identique à celui d'un caractère à forte directionalité correspondant ; la seule différence étant que ces marques ne s'affichent pas.

Algorithme de rendu de base

L'algorithme bidirectionnel lit un flux de texte et exécute trois phases principales :

- Découpe du texte lu en paragraphes. Le reste de l'algorithme s'applique consécutivement à chaque paragraphe.
- Résolution des niveaux d'enchâssement du texte. À cette fin, on utilise au cours de cette phase le type directionnel des caractères ainsi que les codes de formatage explicites.
- Après découpe du texte en lignes, réordonnancement du texte pour l'afficher ligne à ligne en utilisant les niveaux d'enchâssement résolus.

L'algorithme ne réordonne le texte qu'au sein d'un même paragraphe ; les caractères d'un paragraphe n'ont aucun effet sur les caractères d'un autre paragraphe. Les paragraphes sont divisés à l'aide d'un U+2029 SÉPARATEUR DE PARAGRAPHES ou d'une fonction de passage à la

ligne appropriée (voir la section *4.3, Directionalité – normatif*, ainsi que le rapport technique Unicode n° 13, « Unicode Newline Guidelines » présent sur le disque ou la version tenue à jour sur le site Internet du consortium pour plus de renseignements sur la manière de traiter les retours de chariot (*CR*), passages à la ligne (*LF*) et les combinaisons des deux caractères (*CRLF*)). On peut utiliser un protocole de niveau supérieur pour délimiter les paragraphes, ainsi du texte situé dans deux cellules différentes d'un tableau constitue-t-il deux paragraphes séparés.

Les caractères combinatoires suivent toujours leur caractère de base dans la représentation du texte en mémoire. Même après le réordonnancement nécessaire à l'affichage et au formage des caractères, les glyphes qui représentent le caractère combinatoire restent attachés en mémoire au glyphe correspondant à leur caractère de base. Selon l'orientation de la ligne et la directionalité des glyphes correspondant aux lettres de base, ce caractère combinatoire peut s'adjoindre à ce glyphe, par exemple, par la gauche, la droite ou le haut.

Dans les sous-sections suivantes, on distingue les définitions normatives des règles normatives grâce à la numération précisée au tableau 3-6.

Tableau 3-6. Règles et définitions normatives

Numération	Section
BDn	Définitions
Pn	Niveaux de paragraphe
Xn	Niveaux et directions explicites
Fn	Types faibles
Nn	Types neutres
In	Niveaux implicites
Rn	Niveaux résolus

Définitions

BD1 *Type bidirectionnel de caractère* : valeur attribuée à chaque caractère Unicode, y compris les caractères non affectés.

BD2 *Niveau d'enchâssement :* nombre qui indique jusqu'à quel point un texte est imbriqué et la direction implicite du texte à ce niveau. Le niveau d'enchâssement minimum est de zéro, la profondeur explicite maximale est de 61.

- Les niveaux d'enchâssement sont ajustés de manière explicite par les codes de forçage et d'enchâssement ; plus le nombre est élevé, plus le texte est profondément imbriqué. La limite de profondeur est imposée dans le seul but de fournir une limite précise de pile à toutes les mises en œuvre et garantir de la sorte des résultats identiques. Soixante et un niveaux sont plus que suffisants, même lorsque ces niveaux sont générés mécaniquement ; au-delà d'un petit nombre d'enchâssements l'affichage devient vite embrouillé.

BD3 On appelle *direction d'enchâssement* le sens implicite du niveau d'enchâssement courant (pour un caractère donné). Il vaut **G** (gauche) si le niveau d'enchâssement est paire et **D** (droite) si le niveau d'enchâssement est impair.

- Ainsi pour un morceau de texte particulier, le niveau 0 peut-il être du français, le niveau 1 de l'arabe, peut-être imbriqué dans le texte français de niveau 0. Le niveau 2 pourrait être un texte français enchâssé dans le texte arabe de niveau 1 et ainsi de suite. À moins que l'on force leur direction, le texte et les nombres français seront toujours situés à un niveau pair ; le texte arabe (à l'exception des nombres arabes)

sera toujours situé à un niveau impair. La signification précise des niveaux d'enchâssement apparaîtra lors de l'examen de l'algorithme de réordonnancement, bien que la présente section illustre le fonctionnement de cet algorithme.

BD4 Le *niveau d'enchâssement de paragraphe* correspond au niveau d'enchâssement qui précise l'orientation bidirectionnelle implicite du paragraphe en question.

BD5 On appelle *direction de paragraphe* le sens associé au niveau d'enchâssement de paragraphe.

- Dans certains contextes, la direction de paragraphe s'appelle également aussi la direction de base.

BD6 L'*état de forçage directionnel* précise si le type bidirectionnel des caractères est forcé à l'aide d'une commande directionnelle explicite. Cet état a trois valeurs, énumérées au tableau 3-7.

Tableau 3-7. État de forçage directionnel

Valeur	Interprétation
Neutre	Pas de dérogation active
Droite-à-gauche	Les caractères doivent être forcés à D
Gauche-à-droite	Les caractères doivent être forcés à G

BD7 Un *palier* correspond à la sous-chaîne maximale de caractères qui ont le même niveau d'enchâssement. Elle est maximale dans la mesure où aucun caractère qui précède ou suit cette sous-chaîne n'appartient au même niveau.

Exemple. Dans cet exemple, ainsi que les suivants, on utilisera la chasse des caractères pour indiquer leur directionalité implicite. Les lettres majuscules représentent des caractères s'écrivant de droite à gauche (comme en arabe ou en hébreu), alors que les lettres minuscules représentent les caractères s'écrivant de gauche à droite (comme c'est le cas en français ou en russe).

```
Mémoire :                   voiture se dit LA VOITURE en arabe
Directionalité des caractères : GGGGGGG-GG-GGG-DD-DDDDDDD-GG-GGGGG
Niveaux résolus :           00000000000000011111111110000000000
```

On constate que le caractère neutre (l'espace) entre LA et VOITURE appartient au même niveau que les caractères qui l'entourent. C'est de la sorte que les marques directionnelles ponctuelles agissent : l'insertion de marques directionnelles ponctuelles appropriées autour de caractères neutres en modifie le niveau.

Types bidirectionnels de caractères. Les types bidirectionnels de caractères normatifs de tous les caractères Unicode sont précisés dans la Base de données de caractères Unicode ; les différents types sont présentés au tableau 3-8. L'abréviation anglaise des types apparaît en italiques et entre parenthèses, après l'abréviation française. Ces abréviations anglaises se retrouvent dans les fichiers du cédérom, nous avons cependant adopté, pour des raisons pédagogiques et mnémotechniques, des abréviations françaises dans ce chapitre explicatif.

Tableau 3-8. Types bidirectionnels de caractère

Catégorie	Type	Description	Caractères de ce type
Forte	G *(L)*	gauche-à-droite	MGAD (*LRM*), la plupart des caractères alphabétiques, syllabiques, idéographiques han, chiffres qui ne sont ni européens ni arabes, tous les caractères non affectés sauf dans les intervalles (0590-05FF,FB1D-FB4F) et (0600-07BF,FB50-FDFF, FE70-FEFF)
	EGAD (*LRE*)	enchâssement gauche-à-droite	EGAD (*LRE*)
	FGAD *(LRO)*	forçage gauche-à-droite	FGAD *(LRO)*
	D *(R)*	droite-à-gauche	MDAG *(RLM)*, l'alphabet hébreu, la plupart des signes de ponctuation propres à l'hébreu, tous les caractères non affectés dans les intervalles (0590-05FF, FB1D-FB4F)
	DA *(AL)*	droite-à-gauche arabe	Les alphabets arabe, thâna et syriaque, la plupart des signes de ponctuation propres à ces écritures, tous les caractères non affectés dans les intervalles (0600-07BF, FB50-FDFF, FE70-FEFF)
	EDAG *(RLE)*	enchâssement droite-à-gauche	EDAG *(RLE)*
	FDAG *(RLO)*	forçage droite-à-gauche	FDAG *(RLO)*
Faible	DFD *(PDF)*	dépilement de formatage bidirectionnel	DFD *(PDF)*
	NE *(EN)*	nombres européens	Chiffres européens, chiffres arabo-hindî orientaux,…
	SE *(ES)*	séparateur de nombres européens	Barre oblique (cotice)
	TE *(ET)*	terminateur de nombre européen	Signe plus, signe moins, degré, symboles monétaires,…
	NA *(AN)*	nombres arabes	Chiffres arabo-hîndi, séparateur des dizaines et des milliers arabe,…
	SC *(CS)*	séparateur commun de chiffres	Point-virgule, virgule, point final (point), espace insécable,…
	SACN *(NSM)*	signe à chasse nulle	Caractères qualifiés d'un Mn (signe à chasse nulle) ou d'un Me (signe englobant) dans la Base de données de caractères Unicode
	SED *(BN)*	sans effet directionnel[20]	Caractères de formatage et de commande qui ne correspondent à aucun type ci-dessus (ils doivent être ignorés lors du traitement de textes bidirectionnels)

[20] Le terme anglais Boundary neutral (« neutre aux frontières ») décrit une fonction historique de ce type de caractère qui n'est plus pertinente.

Catégorie	Type	Description	Caractères de ce type
Neutre	SP *(B)*	séparateur de paragraphes	Séparateur de paragraphes, fonctions de passage à la ligne adéquates, analyse de paragraphe de protocole de niveau supérieur
	SS *(S)*	séparateur de segments	Tabulation
	BL *(WS)*	blanc	Espace, espacement de lisibilité entre chiffres, séparateur de lignes, changement de page, espaces générales,…
	AN *(ON)*	autres neutres	Tous les autres caractères, y compris le CARACTÈRE DE REMPLACEMENT D'OBJET

- Les terme *nombres* ou *chiffres européens* renvoient aux formes décimales habituelles utilisées en Europe et ailleurs dans le monde. Le terme *chiffres arabo-hindî*, pour sa part, désigne les formes indigènes arabes (voir la section *9.2, Arabe* pour plus de détails sur la dénomination des chiffres).
- On attribue aux numéros de caractères non affectés une directionalité forte. Cette convention est une exception expresse aux exigences de conformité générale d'Unicode relatives aux caractères non affectés. L'affectation ultérieure de caractères à ces numéros pourrait signifier un changement de type bidirectionnel.
- Les mises en œuvre conformes peuvent attribuer aux caractères à usage privé des types bidirectionnels différents.
- Pour l'algorithme bidirectionnel, on considère les objets incorporés (comme les images) de la même manière que le U+FFFC CARACTÈRE DE REMPLACEMENT D'OBJET.

Le tableau 3-9 énumère les abréviations supplémentaires utilisées dans les exemples et les types de caractère internes employés dans cet algorithme.

Table 3-9. Abréviations utilisées dans les exemples et types internes

Symbole	Description
N	Neutre ou séparateur (SP, SS, BL, AN).
i	Directionalité (G ou D) qui correspond à la direction du niveau d'enchâssement (pair ou impair) pour le passage courant. Voir X10 ci-dessous.
dp	Directionalité (G ou D) attribuée à la position qui précède un palier.
fp	Directionalité (G ou D) attribuée à la position qui suit un palier.

Résolution des niveaux d'enchâssement

Le corps de l'algorithme bidirectionnel fait appel aux types bidirectionnels des caractères et aux codes explicites pour produire une liste de niveaux d'enchâssement résolus. Cette résolution comprend cinq étapes : (1) établissement du niveau de paragraphe, (2) établissement des niveaux d'enchâssement et des directions explicites, (3) résolution des types faibles, (4) résolution des types neutres et (5) résolution des niveaux d'enchâssement implicites.

Niveau de paragraphe. L'algorithme bidirectionnel s'applique aux paragraphes.

P1 *Découper le texte en paragraphes. Un séparateur de paragraphes accompagne le paragraphe qui le précède. Appliquer toutes les autres règles de cet algorithme à chaque paragraphe.*

P2 *Pour chaque paragraphe, trouver le premier caractère à forte directionalité (G, DA, D)*[21]*.*

Étant donné que les séparateurs de paragraphes délimitent le texte dans cet algorithme, un caractère fort , en règle générale, suivra un séparateur de paragraphes ou se trouvera au tout début du texte.

P3 *Si le caractère obtenu à la règle P2 est de type DA ou D alors initialiser le niveau d'enchâssement à 1, sinon à zéro.*

Remarquons que lorsqu'un protocole de niveau supérieur précise le niveau de paragraphe, il n'est pas nécessaire d'appliquer les règles P2 et P3.

Niveaux et directions explicites. On détermine les niveaux d'enchâssement explicites à l'aide des codes d'enchâssement et de forçage, par l'application des règles de niveaux explicites X1 à X9. On applique ces règles lors du même passage logique sur les données en entrée.

Enchâssements explicites. Un code d'enchâssement explicite fixe le niveau d'enchâssement du texte, mais il ne change pas la directionalité de chacun des caractères impliqués (contrairement aux codes de forçage).

X1 *Initialiser tout d'abord le niveau d'enchâssement courant au niveau d'enchâssement du paragraphe. Initialiser l'état de forçage bidirectionnel à neutre. Traiter chaque caractère de manière itérative en appliquant les règles X2 à X9. Seuls les niveaux d'enchâssement de 0 à 61 sont valables pour cette phase.*

Il se peut que le niveau d'enchâssement maximal de 62 soit atteint lors de la résolution des niveaux par les règles I1 et I2.

X2 *Pour chaque EDAG, calculer le plus petit niveau d'enchâssement impair supérieur* au niveau courant.

a. *Si ce nouveau niveau est valable alors ce code d'enchâssement est valable. Se souvenir (empiler) du niveau d'enchâssement et de l'état de forçage courants. Réinitialiser le niveau courant à ce nouveau niveau et l'état de forçage à* neutre.

b. *Si ce nouveau niveau n'est pas valable, alors ce code n'est pas valable. Ne changer ni le niveau d'enchâssement ni l'état de forçage courants.*

Exemple, niveau 0 → 1 ; niveaux 1, 2 → 3 ; niveaux 3, 4 → 5 ; …59,60 → 61 ; au-delà de 60, aucun changement (ne pas changer de niveaux si le nouveau niveau n'est pas valable).

X3 Pour chaque EGAD, calculer le plus petit niveau d'enchâssement pair supérieur au niveau courant.

[21] Et uniquement ces valeurs (G, DA, D) et non tous les caractères à forte directionalité du tableau 3-7.

[22] On parle également de code d'enchâssement explicite.

a. Si ce nouveau niveau est valable alors ce code d'enchâssement est valable. Se souvenir (empiler) du niveau d'enchâssement et de l'état de forçage courants. Réinitialiser le niveau courant à ce nouveau niveau et l'état de forçage à neutre*.*

b. Si ce nouveau niveau n'est pas valable, alors ce code n'est pas valable. Ne changer ni le niveau d'enchâssement ni l'état de forçage courants.

Par exemple, niveau 0, 1 → 2 ; niveaux 2, 3 → 4 ; niveaux 4, 5 → 6 ; …58, 59 → 60 ; au-delà de 59, aucun changement (ne pas changer de niveaux si le nouveau niveau n'est pas valable.)

Forçages explicites. Un forçage directionnel explicite ajuste le niveau d'enchâssement de la même manière qu'un code d'enchâssement explicite, cependant il impose également sa directionalité à tous les caractères impliqués.

X4 Pour chaque FDAG, calculer le plus petit niveau d'enchâssement impair *supérieur* au niveau courant*.*

a. Si ce nouveau niveau est valable alors ce code d'enchâssement est valable. Se souvenir (empiler) du niveau d'enchâssement et de l'état de forçage courants. Affecter au niveau courant la valeur de ce nouveau niveau et mettre l'état de forçage à droite-à-gauche*.*

b. Si ce nouveau niveau n'est pas valable, alors ce code n'est pas valable. Ne changer ni le niveau d'enchâssement ni l'état de forçage courants.

X5 Pour chaque FGAD, calculer le plus petit niveau d'enchâssement pair *supérieur* au niveau courant*.*

a. Si ce nouveau niveau est valable alors ce code d'enchâssement est valable. Se souvenir (empiler) du niveau d'enchâssement et de l'état de forçage courants. Réinitialiser le niveau courant à ce nouveau niveau et l'état de forçage à gauche-à-droite*.*

b. Si ce nouveau niveau n'est pas valable, alors ce code n'est pas valable. Ne changer ni le niveau d'enchâssement ni l'état de forçage courants.

X6 Pour tous les types sauf EDAG, EGAD, FDAG, FGAD et DFD :

a. Ajuster le niveau du caractère courant au niveau d'enchâssement courant.

b. Si l'état de forçage directionnel n'est pas neutre, réinitialiser le type bidirectionnel du caractère courant à l'état de forçage directionnel.

Si l'état de forçage directionnel est neutre alors les caractères conservent leurs types habituels : les caractères arabes restent DA, les caractères latins demeurent L, les neutres N et ainsi de suite. Si l'état de forçage directionnel est égal à D alors les caractères deviennent D. Si l'état de forçage est égal à G, les caractères passent alors à G.

Fin des enchâssements et des forçages. Un même code met fin à l'action des deux types de codes à effet prolongé, qu'il s'agisse d'un enchâssement ou d'un forçage directionnel. Tous les codes et états empilés sont dépilés à la fin de chaque paragraphe.

X7 Pour chaque DFD, trouver le code d'enchâssement ou de forçage correspondant. Si un code correspondant valable existe, rétablir (dépiler) les derniers niveau d'enchâssement et état de forçage directionnel mémorisés (empilés).

X8 *On met un terme à l'action de tous les codes à effet prolongé (enchâssement et forçage) à la fin de chaque paragraphe. Les séparateurs de paragraphes* ne font pas *partie de l'enchâssement.*

X9 *Éliminer tous les codes EDAG, EGAD, FDAG, FGAD, DFD et SED.*

- Il n'est pas nécessaire que les mises en œuvre éliminent réellement ces codes, il leur suffit de se comporter pour le reste de l'algorithme comme si ces codes n'existaient plus. La conformité à Unicode n'impose aucun emplacement particulier à ces codes, pour peu que tous les autres caractères soient correctement ordonnés.

Voir les « *Notes de mise en œuvre »* plus loin dans cette section pour l'information concernant la mise en œuvre de l'algorithme sans suppression des codes de formatage.

X10 *On applique les règles restantes à chaque passage de caractères situé au même niveau, appelé palier. Pour chaque passage, établir la directionalité du début-de-palier* (dp) *et de la fin-de-palier* (fp)*, G ou D. Cette valeur dépend du plus haut des deux niveaux de chaque côté de la limite (au début ou à la fin du paragraphe le niveau de l'« autre » passage est le niveau d'enchâssement de base). Si le niveau supérieur est impair, la directionalité est D, sinon elle est G.*

Exemple :

Niveaux : 0 0 0 1 1 1 2
Passages : ← 1 → ← 2 → <3>

Le passage 1 est au niveau 0, *dp* égale G, *fp* égale D.
Le passage 2 est au niveau 1, *dp* égale D, *fp* égale G.
Le passage 3 est au niveau 2, *dp* égale G, *fp* égale G.

Pour deux passages contigus, le *fp* du premier passage est égal au *dp* du second.

Résolution des types faibles. On peut maintenant résoudre les types faibles, palier par palier. Aux limites de palier, quand il faut connaître le type du caractère situé de l'autre côté de la limite, on utilise la directionalité affectée au *fp* ou au *dp*.

On peut maintenant résoudre les signes sans chasse grâce aux caractères précédents.

W1 *Vérifier chaque signe à chasse nulle (SACN) du palier et modifier leur type de SACN à celui du caractère précédent. Si le SACN se situe en début de palier, il prend la directionalité du* dp*.*

Supposons que, dans cet exemple, *dp* soit D :

DA SACN SACN → DA DA DA
dp *SACN* → dp D

On analyse ensuite tout le texte à la recherche des nombres. Cette passe modifiera les types directionnels suivants : *séparateur de chiffres européens*, *terminateur de chiffres européens* et *séparateur de chiffres communs* en *chiffres européens, chiffres arabes* ou *autres neutres*. Des morceaux du texte à analyser ont peut-être déjà vu leurs types modifiés par des forçages directionnels. Si c'est le cas, ceux-ci ne seront plus considérés comme des nombres.

W2 *Remonter en amont de chaque occurrence d'un nombre européen jusqu'à trouver le premier type fort (D,G, DA ou* dp*). Si on trouve un DA, changer le type de nombres européens à nombres arabes.*

DA NE → DA NA

DA N NE → DA N NA
dp N NE → *dp* N NE
G N NE → G N NE
D N NE → D N NE

W3 *Changer tous les DA en D.*

W4 *Un* séparateur européen *entre deux* nombres européens *se change en un* nombre européen*. Un* séparateur commun *entre deux chiffres de même type, prend ce type :*

NE ES NE → NE NE NE
NE CS NE → NE NE NE
NA CS NA → NA NA NA

W5 *Une suite de* terminateurs européens *contigüe à des* nombres européens *devient une série de nombres* européens :

ET ET NE → NE NE NE
NE ET ET → NE NE NE
NA ET NE → NA NE NE

W6 *Sinon, les séparateurs et terminateurs se changent en* autres neutres *:*

NA ET → NA AN
G ES NE → G AN NE
NE CS NA → NE AN NA
ET NA → AN NA

W7 *Remonter en amont de chaque occurrence d'un nombre européen jusqu'à trouver le premier type fort (D,G ou* dp*). Si on trouve un G, changer le type de* chiffres européens *à G.*

G N NE → G N G
D N NE → D N NE

Résolution des types neutres. On peut maintenant résoudre les types neutres, palier par palier. Aux limites de palier, quand il faut connaître le type du caractère situé de l'autre côté de la limite, on utilise la directionalité affectée à *fp* ou à *dp*.

La phase suivante détermine la direction des neutres. À la fin de cette phase, tous les neutres sont devenus **G** ou **D**. En règle générale, les neutres prennent la direction du texte qui les jouxte. En cas de conflit, ils adoptent la direction de l'enchâssement.

N1 *Une suite de neutres adopte la direction du texte fort contigu si le texte des deux côtés a la même direction. Les* chiffres européens *et* arabes *sont considérés comme du texte D. On utilise un début-de-palier* (dp) *et une fin-de-palier* (fp) *pour délimiter les paliers.*

D N D → D D D
G N G → G G G
D N NA → D D NA
NA N D → NA D D
D N NE → D D NE
NE N D → NE D D
NE N NA → NE D NA

- À ce stade, tous les chiffres européens après un G ont déjà été convertis en G, seuls les chiffres européens après D sont donc traités comme des D.

N2 *Tous les neutres restants adoptent la direction de l'enchâssement.*

N → i

Supposons dans cet exemple que *fp* soit égal à G et *dp* à D :

```
G   N   fp  → G   G   fp
D   N   fp  → D   i   fp
dp  N   G   → fp  i   G
dp  N   D   → fp  D   D
```

Exemples. Une liste de nombres séparés par des neutres et enchâssés dans un passage directionnel sera affichée dans l'ordre de ce passage.

Mémoire : il dit « LES VALEURS SONT 123, 456, 789, ÇA VA ».
Affichage : il dit « AV AÇ ,789 ,456 ,123 TNOS SRUELAV SEL ».

Dans ce cas, à la fois les virgules et les espaces entre les nombres adoptent la direction du texte avoisinant (majuscules = droite-à-gauche) tout en ignorant les nombres. On ne considère pas que les virgules fassent partie des nombres car elles ne sont pas entourées des deux côtés par des nombres. Si une suite gauche-à-droite contigüe existe alors les chiffres européens adopteront cette direction.

Mémoire : il dit « C'EST UNE bmw 500, ÇA VA. »
Affichage : il dit « .AV AÇ ,bmw 500 ENU TSE'C ».

Résolution des niveaux implicites. Dans la dernière phase, il se peut que le niveau d'enchâssement du texte soit augmenté, suite à la directionalité déterminée. Un texte droite-à-gauche aura toujours en fin de compte un niveau impair et les textes gauche-à-droite ou composés de nombres aboutiront toujours à un niveau pair. En outre, les nombres auront toujours en définitive un niveau supérieur à celui du paragraphe. (Il est possible qu'à la suite de ce processus le texte aboutisse à des niveaux supérieurs à 61.) Cette situation donne lieu aux règles suivantes :

I1 *Pour tous les caractères ayant une direction d'enchâssement paire (gauche-à-droite), ceux de type D montent d'un niveau, ceux de type NA ou NE montent de deux niveaux.*

I2 *Pour tous les caractères ayant une direction d'enchâssement impaire (droite-à-gauche), ceux de type G, NA ou NE montent d'un niveau.*

Le tableau 3-10 résume le résultat de l'algorithme implicite de résolution des niveaux.

Tableau 3-10. Résolution des niveaux implicites

Type	Niveau d'enchâssement	
	Pair	Impair
G	NI	NI+1
D	NI+1	NI
NA	NI+2	NI+1
NE	NI+2	NI+1

Remise en ordre des niveaux résolus

L'algorithme ci-dessous décrit le processus logique qui permet d'établir l'ordre d'affichage correct. Comme on l'a fait remarquer plus tôt, ce processus ne décrit pas nécessairement la mise en œuvre réelle qui, pour des raisons d'efficacité, peut diverger de la description logique pour autant qu'elle produise le même résultat. Contrairement aux phases de résolution, cet algorithme opère ligne par ligne *après avoir découpé un paragraphe en lignes pour l'affichage*.

Le processus de découpe d'un paragraphe en une ou plusieurs lignes afin que celles-ci rentrent à l'intérieur de certaines limites ne dépend pas de l'algorithme bidirectionnel. Quand une contextualisation des caractères s'impose, le processus de découpe en lignes peut être quelque peu plus complexe (voir la section *9.2, Arabe*). Il comprend les étapes logiques suivantes :

- On calcule les niveaux du texte selon l'algorithme bidirectionnel.
- On contextualise les caractères pour les transformer en glyphes (*en prenant en compte les niveaux d'enchâssement pour l'effet miroir !*).
- On accumule les largeurs de ces glyphes (*dans l'ordre logique*) pour établir les points de coupure des lignes.
- Pour chaque ligne, on utilise les règles L1-L4 afin de réordonner les caractères sur cette ligne.
- On affiche les glyphes correspondant aux caractères de la ligne dans cet ordre.

L1 *Pour chaque ligne, réinitialiser le niveau d'enchâssement des caractères suivants au niveau d'enchâssement du paragraphe :*

1. *les séparateurs de segments ;*
2. *les séparateurs de paragraphes ;*
3. *toute suite de blancs qui précède un séparateur de segments ou de paragraphes ;*
4. *toute suite de blancs à la fin de la ligne.*

- On utilise ici les types bidirectionnels d'origine et non ceux modifiés par la phase précédente.
- Puisque les séparateurs de paragraphes mettent fin aux lignes, il y en aura au plus un par ligne, à la fin de cette ligne.

Cette exigence, combinée à la règle suivante, signifie que les blancs de fin de ligne doivent apparaître à la fin de la ligne visuelle (dans la direction du paragraphe). Les tabulations doivent toujours avoir une direction cohérente au sein d'un paragraphe.

L2 *Du plus haut niveau présent dans le texte au plus bas niveau impair de chaque ligne, inverser toute suite contigüe de caractères qui est à ce niveau ou plus haut.*

Ce processus inverse une série progressivement plus grande de sous-chaînes. Les quatre exemples ci-dessous illustrent cette règle :

```
Mémoire :           auto se dit AUTO.
Niveaux résolus :   00000000000011110
Niveau inversé 1 :  auto se dit OTUA.

Mémoire :           auto SE DIT AUTO.
```

```
Niveaux résolus :      22221111111111111
Niveau inversé 2 :     otua SE DIT AUTO.
Niveau inversé 1,2 :   .OTUA TID ES auto

Mémoire :              je dis « auto SE DIT AUTO. »
Niveaux résolus :      00000000022221111111111111000
Niveau inversé 2 :     je dis « otua SE DIT AUTO. »
Niveau inversé 1,2 :   je dis « OTUA TID ES auto. »


Mémoire :              DIS-TU QUE « je dis « auto SE DIT AUTO » » ?
Niveaux résolus :      1111111111112222222223444433333333333322111
Niveau inversé 4 :     DIS-TU QUE « je dis « otua SE DIT AUTO » » ?
Niveau inversé 3, 4 :  DIS-TU QUE « je dis « OTUA TID ES auto » » ?
Niveau inversé 2-4 :   DIS-TU QUE « « otua SE DIT AUTO » sid ej » ?
Niveau inversé 1-4 :   ? « je dis « OTUA TID ES auto » » EUQ UT-SID
```

L3 *Les signes combinatoires adjoints à un caractère de base droite-à-gauche doivent, à ce stade, précéder leur caractère de base. Si le moteur de rendu s'attend à ce qu'ils suivent les caractères de base lors du processus final d'affichage, il faut alors inverser les signes combinatoires et les caractères de base.*

De nombreuses fonderies fournissent des signes combinatoires aux paramètres implicites qui permettent d'afficher ces signes par simple saillie. Or, étant donné la remise en ordre pour les caractères droite-à-gauche, il est habituel de faire saillir à gauche les glyphes de la plupart des caractères combinatoires (supposant de la sorte que ces caractères sont adjoints à des caractères de base gauche-à-droite) alors qu'on fait saillir à droite les glyphes des caractères combinatoires appartenant aux écritures droite-à-gauche (supposant de la sorte que ces caractères ne seront adjoints qu'à des caractères de base droite-à-gauche). Avec ces polices, la mise en ordre des glyphes correspondant aux signes combinatoires et aux caractères de base qui précède l'affichage nécessitera peut-être l'ajustement des signes combinatoires quand ceux-ci sont adjoints à des caractères de base « non assortis ». Voir la section *5.14, Rendu des signes à chasse nulle*, pour plus de renseignements.

L4 Un caractère qui possède une propriété miroir selon la section *4.7, Caractères miroirs – normatif*, doit être affichée par le glyphe réfléchi si la directionalité calculée de ce caractère est D.

Ainsi, 0028 PARENTHÈSE GAUCHE — qui est interprétée dans le standard Unicode comme une parenthèse ouvrante — apparaît-elle comme « (» quand le niveau résolu est pair, et comme le glyphe réfléchi «) » quand le niveau résolu est impair.

Conformité bidirectionnelle

L'algorithme bidirectionnel précise une partie de la sémantique intrinsèque aux caractères droite-à-gauche. En l'absence d'un protocole de niveau supérieur qui remplace l'interprétation de la directionalité définie ci-dessus, tout système qui interprète ces caractères doit obtenir, pour être conforme, au rendu un résultat identique à celui produit par l'algorithme bidirectionnel implicite.

Sans effet directionnel. On étiquette un caractère de formatage ou de commande comme SED pour s'assurer qu'il n'a aucun effet sur l'algorithme. Étant donné que l'ordre précis des caractères de formatage par rapport aux autres caractères n'est pas une exigence de conformité, les mises en œuvre peuvent, pour des raisons d'efficacité, les traiter comme bon leur semble pour autant que l'ordre des autres caractères soit conservé.

Codes de formatage à effet prolongé. Comme pour tous les autres caractères Unicode, les systèmes ne sont pas tenus de prendre en charge les codes de formatage à effet prolongé (bien qu'il soit inutile, en règle générale, d'inclure un code terminateur sans inclure son code d'amorce correspondant). D'ordinaire, un système conforme appartient à une des trois classes suivantes :

- *Aucun formatage bidirectionnel.* Ce choix signifie que le système n'interprète pas graphiquement les caractères appartenant aux écritures droite-à-gauche.
- *Bidirectionalité implicite.* Le système prend en charge l'algorithme bidirectionnel implicite ainsi que les marques directionnelles ponctuelles MGAD et MDAG.
- *Pleine bidirectionalité*. Le système prend en charge l'algorithme bidirectionnel implicite, les marques directionnelles ponctuelles (MGAD et MDGA) et les codes d'enchâssement et de forçage directionnels (FGAD, FDAG, EGAD, EDAG, DFD).

Protocoles de niveau supérieur. On retrouve ci-dessous différentes utilisations potentielles d'un protocole de niveau supérieur qu'un système peut adopter pour ordonner des textes bidirectionnels :

- *Forcer le niveau d'enchâssement de paragraphe.* Un protocole de niveau supérieur devrait permettre de changer le niveau d'enchâssement de paragraphe ; que cela soit au niveau des cellules d'un tableau, d'un paragraphe, d'un document ou du système.
- *Changer le traitement des nombres afin d'utiliser des renseignements provenant d'un contexte plus général.* L'analyse des autres paragraphes d'un document peut, par exemple, permettre de déduire que le document est essentiellement en arabe et que les NE devraient en général être converties en des NA.
- *Remplacer, compléter ou neutraliser les codes de forçage et d'enchâssement directionnels.* À cet effet, on peut fournir à l'aide d'une feuille de style ou de balisage des informations relatives au niveau d'enchâssement ou de la directionalité du texte. L'interprétation de cette information doit toujours se faire en fonction du comportement des codes - équivalents mentionnés dans l'algorithme ci-dessus.
- *Modifier les types bidirectionnels attribués aux codes de commande pour qu'ils correspondent à l'interprétation des* codes *de commandes au sein du protocole. (Voir la section 14.1, Codes de commande*.)
- *Substituer les glyphes de chiffres pour les faire correspondre à celles d'un autre jeu*. On peut, par exemple, substituer les glyphes de chiffres arabes pour qu'ils aient la même apparence que les chiffres européens.

Lors de la conversion d'un texte qui utilise un protocole de niveau supérieur, il est nécessaire d'insérer des codes de formatage afin que l'ordre d'affichage corresponde à celui du protocole de niveau supérieur ou (comme dans le dernier exemple) de des caractères appropriés pour que les chiffres apparaissent correctement.

Notes de mise en œuvre

Mise en œuvre de référence. Le code source de deux mises en œuvre de référence de l'algorithme bidirectionnel se trouve dans le rapport technique d'Unicode n° 9, *The Bidirectional Algorithm*, disponible sur le disque qui accompagne cet ouvrage ainsi que, tenu à jour, sur le site Internet du consortium Unicode. On recommande aux concepteurs d'utiliser ces ressources pour tester leurs mises en œuvre.

Conservation des codes de formatage. Certaines mises en œuvre pourraient vouloir conserver les codes de formatage lors de l'exécution de l'algorithme. Pour ce faire, il faut :

- À la règle X9, plutôt que d'éliminer les codes de formatage, ajuster tous les codes de formatage à SED. Pour chaque suite de SED, régler le niveau de ces codes à celui du code non-SED précédent (ou si, cette suite se situe au début du paragraphe, au niveau du paragraphe).
- À la règle W1, rechercher en amont de chaque SACN le premier caractère du palier dont le type n'est pas SED, modifier le type du SACN à celui du caractère trouvé.
- À la règle W4, ignorer lors de l'analyse les types SED qui jouxtent des SE ou des NE.
- À la règle W5, modifier toutes les suites adéquates de TE et de SED et non pas simplement les TE.
- À la règle W6, changer également tous les types SED en AN.
- À la règle L1, inclure les codes de formatage et les SED ainsi que les blancs parmi les suites dont les niveaux sont réinitialisés avant un séparateur ou une coupure de ligne.

Les mises en œuvre qui affichent de manière lisible les caractères de formatage devront sans doute modifier ce mécanisme afin de positionner les caractères de formatage de façon optimale pour l'édition.

Texte vertical. On utilise l'algorithme bidirectionnel pour calculer les niveaux directionnels d'un texte, même dans les cas où les lignes s'écrivent verticalement. Néanmoins, on n'utilise pas ces niveaux pour réordonner le texte, car les caractères sont habituellement ordonnés de manière uniforme de haut en bas. Ces niveaux servent plutôt à établir la rotation du texte. Il arrive parfois que les lignes verticales suivent une ligne de base verticale sur laquelle les caractères sont orientés normalement (sans rotation), les caractères y sont ordonnés de haut en bas qu'il s'agisse d'hébreu, de chiffres ou de caractères latins. Cependant, lors de la composition verticale de textes écrits en écriture arabe, il est plus fréquent d'utiliser une ligne de base horizontale que l'on fait pivoter de 90° en sens anti-horaire, de telle sorte que les caractères soient dans le bon ordre de haut en bas. On peut faire pivoter de 90° en sens horaire le texte latin ou les chiffres afin qu'eux aussi se lisent dans l'ordre de haut en bas.

L'algorithme bidirectionnel rentre également en jeu quand certains caractères sont ordonnés de bas en haut. Cette situation se produit, par exemple, quand on mélange des glyphes arabes et latins et que l'on fait pivoter tous les glyphes de 90° en sens horaire. (La décision de présenter un texte horizontalement ou verticalement ou de faire pivoter le texte ne relève pas du standard Unicode mais de protocoles de niveau supérieur.)

Utilisation. Grâce aux types de caractères implicites, à l'heuristique de résolution des neutres et au comportement bidirectionnel des chiffres, l'algorithme de mise en ordre bidirectionnel implicite produit habituellement l'affichage escompté sans aucun traitement particulier supplémentaire. Il surgit parfois, cependant, des cas problématiques quand un paragraphe droite-à-gauche commence avec des caractères gauche-à-droite ou dans les cas de segments de texte imbriqués et de directionalités différentes ou, enfin, quand des caractères faibles bordent les frontières directionnelles. Dans ces cas-là, des marques de directionalité ou d'enchâssement peuvent s'avérer nécessaires pour l'obtention d'un rendu correct. Les numéros de pièce ou de série nécessitent parfois aussi l'utilisation de forçages directionnels.

Le problème le plus courant résulte de l'utilisation de neutres à la frontière d'une langue imbriquée. On règle le problème en réglant le niveau du texte imbriqué à la bonne valeur. Ainsi, avec l'ensemble du texte au niveau 0, on obtient :

Mémoire : `il dit « J'AI SOIF! » et expira.`

Affichage : `il dit « FIOS IA'J! » et expira.`

Si le point d'exclamation doit faire partie de la citation arabe, l'utilisateur peut alors sélectionner le texte J'AI SOIF! et l'enchâsser de manière explicite comme un texte arabe pour produire le résultat suivant :

Mémoire : `il dit « <EDAG>J'AI SOIF!<DFD> » et expira.`

Affichage : `il dit « !FIOS IA'J » et expira.`

Une méthode plus simple consiste à placer une marque directionnelle droite après le point d'exclamation. Le point d'exclamation ne se trouvant alors plus sur une frontière directionnelle, le résultat obtenu est le bon.

On préfère cette dernière méthode, car elle n'implique pas l'utilisation de codes de formatage à états qui peuvent facilement se désynchroniser si les éditeurs et autres processus de manipulation de chaînes ne les gèrent pas pleinement[23]. On n'utilise habituellement les codes de formatage à états que dans les cas plus complexes (et plus rares) de double enchâssement (voir ci-dessous).

Mémoire : `DISIEZ-VOUS «<EGAD> il dit « J'AI SOIF!<MDAG> » et expira. <DFD> » ?`

Affichage : `? « il dit « !FIOS IA'J » et expira. » SOUV-ZEISID`

Migration de 2.0 à 3.0. La version 3.0 de la Base de données de caractères Unicode introduit de nouveaux types de caractère bidirectionnel de sorte que l'algorithme ne dépend désormais plus que de types de caractère et non plus de certaines valeurs de caractères. Les modifications apportées aux types bidirectionnels de la version 2.0 sont répertoriées au tableau 3-11.

Tableau 3-11. Nouveaux types bidirectionnels

Caractères	Nouveau type bidirectionnel
Tous les caractères des catégories générales Me, Mn	SACN
Tous les caractères de type D dans les intervalles arabes (0600..06FF, FB50..FDFF, FE70..FEFE) Les lettres dans les intervalles thâna et syriaque ont également cette valeur.	DA
Les caractères d'enchâssement et de forçage explicite : FGAD (*LRO*), FDAG (*RLO*), EGAD (*LRE*), EDAG (*RLE*), DFD (*PDF*).	FGAD, FDAG, EGAD, EDAG, DFD, respectivement
Caractères de formatage et de commande (catégorie générale Cf et Cc) qui étaient de type bidirectionnel AN	SED
Espace sans chasse	SED

Les mises en œuvre qui utilisent les anciens tableaux de propriétés peuvent être rectifiées pour prendre en compte les modifications à l'algorithme bidirectionnel en associant de manière automatique aux caractères répertoriés au tableau 3-11 leurs nouveaux types.

[23] Par exemple, lors d'un copier-coller.