Union internationale des télécommunications

UIT-T

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

X.891

(05/2005)

SÉRIE X: RÉSEAUX DE DONNÉES, COMMUNICATION ENTRE SYSTÈMES OUVERTS ET SÉCURITÉ

Applications OSI – Applications génériques de l'ASN.1

Technologies de l'information – Applications génériques de l'ASN.1: Fast Infoset

Recommandation UIT-T X.891

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX DE DONNÉES, COMMUNICATION ENTRE SYSTÈMES OUVERTS ET SÉCURITÉ

RÉSEAUX PUBLICS DE DONNÉES	
Services et fonctionnalités	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalisation et commutation	X.50–X.89
Aspects réseau	X.90–X.149
Maintenance	X.150–X.179
Dispositions administratives	X.180–X.199
INTERCONNEXION DES SYSTÈMES OUVERTS	
Modèle et notation	X.200–X.209
Définitions des services	X.210–X.219
Spécifications des protocoles en mode connexion	X.220–X.229
Spécifications des protocoles en mode sans connexion	X.230–X.239
Formulaires PICS	X.240–X.259
Identification des protocoles	X.260–X.269
Protocoles de sécurité	X.270–X.279
Objets gérés des couches	X.280–X.289
Tests de conformité	X.290–X.299
INTERFONCTIONNEMENT DES RÉSEAUX	
Généralités	X.300–X.349
Systèmes de transmission de données par satellite	X.350–X.369
Réseaux à protocole Internet	X.370–X.379
SYSTÈMES DE MESSAGERIE	X.400–X.499
ANNUAIRE	X.500–X.599
RÉSEAUTAGE OSI ET ASPECTS SYSTÈMES	
Réseautage	X.600–X.629
Efficacité	X.630–X.639
Qualité de service	X.640–X.649
Dénomination, adressage et enregistrement	X.650–X.679
Notation de syntaxe abstraite numéro un (ASN.1)	X.680–X.699
GESTION OSI	
Cadre général et architecture de la gestion-systèmes	X.700–X.709
Service et protocole de communication de gestion	X.710–X.719
Structure de l'information de gestion	X.720–X.729
Fonctions de gestion et fonctions ODMA	X.730–X.799
SÉCURITÉ	X.800–X.849
APPLICATIONS OSI	
Engagement, concomitance et rétablissement	X.850–X.859
Traitement transactionnel	X.860–X.879
Opérations distantes	X.880–X.889
Applications génériques de l'ASN.1	**X.890–X.899**
TRAITEMENT RÉPARTI OUVERT	X.900–X.999
SÉCURITÉ DES TÉLÉCOMMUNICATIONS	X.1000–

Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.

NORME INTERNATIONALE ISO/CEI 24824-1
RECOMMANDATION UIT-T X.891

Technologies de l'information – Applications génériques de l'ASN.1: Fast Infoset

Résumé

La présente Recommandation | Norme internationale spécifie une représentation d'une instance de l'ensemble d'informations du langage XML du W3C qui utilise des codages binaires. Ces codages binaires sont spécifiés à l'aide de la notation ASN.1 et de la notation de commande de codage ASN.1 (ECN, *encoding control notation*).

La technologie spécifiée dans la présente Recommandation | Norme internationale se nomme Fast Infoset. Elle fournit une alternative à la syntaxe XML du W3C pour représenter des instances de l'ensemble d'informations du langage XML du W3C. Cette représentation donne généralement des tailles de codage plus petites et un traitement plus rapide que la représentation XML W3C.

La présente Recommandation | Norme internationale spécifie l'utilisation de plusieurs techniques qui minimisent la taille des codages (appelés documents Fast Infoset) et qui maximisent la vitesse de création et de traitement de documents Fast Infoset. Ces techniques incluent l'utilisation de tableaux dynamiques (pour les chaînes de caractères et pour les noms qualifiés), de vocabulaires initiaux, et de vocabulaires externes.

La présente Recommandation | Norme internationale spécifie aussi un type de support d'extensions de messagerie Internet multiobjets (MIME, *multipurpose Internet mail extensions*) qui identifie un document Fast Infoset.

Source

La Recommandation UIT-T X.891 a été approuvée le 14 mai 2005 par la Commission d'études 17 (2005-2008) de l'UIT-T selon la procédure définie dans la Recommandation UIT-T A.8. Elle inclut les corrections apportées par le Corrigendum 1 de la Recommandation UIT-T X.891 (2005) approuvé le 13 juin 2006 par la Commission d'études 17 (2005-2008) de l'UIT-T selon la procédure définie dans la Recommandation UIT-T A.8. Un texte identique est publié comme Norme Internationale ISO/CEI 24824-1.

AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de l'AMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

Le respect de cette Recommandation se fait à titre volontaire. Cependant, il se peut que la Recommandation contienne certaines dispositions obligatoires (pour assurer, par exemple, l'interopérabilité et l'applicabilité) et considère que la Recommandation est respectée lorsque toutes ces dispositions sont observées. Le futur d'obligation et les autres moyens d'expression de l'obligation comme le verbe "devoir" ainsi que leurs formes négatives servent à énoncer des prescriptions. L'utilisation de ces formes ne signifie pas qu'il est obligatoire de respecter la Recommandation.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT avait été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

TABLE DES MATIÈRES

Introduction

La présente Recommandation | Norme internationale spécifie une représentation d'une instance de l'ensemble d'informations XML W3C qui utilise des codages binaires (spécifiés à l'aide de la notation ASN.1 et de la notation de contrôle de codage ASN.1). Le codage spécifié dans l'édition de la présente Recommandation | Norme internationale est identifié par le numéro de version 1 (voir le § 12.9).

La technologie spécifiée dans la présente Recommandation | Norme internationale se nomme Fast Infoset. Elle fournit une alternative à la syntaxe XML W3C comme moyen de représenter les instances d'ensemble d'informations XML du W3C. Cette représentation donne habituellement des tailles de codage plus petites et un traitement plus rapide que la représentation XML du W3C.

La représentation d'une instance de l'ensemble d'informations XML W3C spécifiée dans la présente Recommandation | Norme internationale s'appelle un document Fast Infoset. Chaque document Fast Infoset est un codage d'une valeur abstraite d'un type de données ASN.1 (le type **`Document`** – voir le § 7.2) représentant une instance de l'ensemble d'informations XML W3C.

La présente Recommandation | Norme internationale spécifie l'utilisation de plusieurs techniques qui minimisent la taille d'un document Fast Infoset et qui maximisent la vitesse de création et de traitement de tels documents.

Ces techniques sont fondées sur l'emploi de tableaux de vocabulaire, qui permettent d'utiliser des valeurs entières normalement faibles (les indices de tableaux de vocabulaire) à la place des chaînes de caractères qui forment (par exemple) les noms des éléments ou attributs dans une mise en série XML 1.0 d'une instance de l'ensemble d'informations XML W3C.

Il y a de nombreux tableaux de vocabulaire (voir le paragraphe 8), dont les plus fondamentaux (les tableaux de chaînes de huit caractères) transposent des entiers normalement petits en chaînes de caractères. Il y a cependant aussi des tableaux de vocabulaire (le tableau des noms d'éléments et le tableau des noms d'attributs) qui donnent un niveau d'indication supplémentaire, avec un indice de tableau de vocabulaire transposant un ensemble de trois indices de tableaux de vocabulaire, identifiant un préfixe, un nom d'espace de nom, et un nom local.

Une autre technique importante est l'utilisation d'un tableau de vocabulaire à alphabet restreint. Il contient des entrées qui font la liste d'un sous-ensemble de caractères ISO/CEI 10646. Si une chaîne de caractères qui doit être codée possède une entrée dans ce tableau, elle peut alors être codée en notant qu'on utilise ce tableau de vocabulaire, en donnant l'indice du tableau de vocabulaire, et en codant alors chaque caractère avec le nombre minimal de bits nécessaires pour ce sous-ensemble particulier de caractères ISO/CEI 10646. Il y a un certain nombre d'alphabets restreints préconstruits qui forment toujours les premières entrées de ce tableau, couvrant les chaînes qu'on rencontre de façon usuelle comme les dates, les heures et les valeurs numériques.

Une importante technique d'optimisation utilise le tableau de vocabulaire d'algorithme de codage. Ce tableau identifie les codages spécialisés qui peuvent être employés pour les chaînes d'utilisation courante, toujours avec un certain nombre d'algorithmes incorporés. Par exemple, s'il y a une chaîne qui ressemble à la représentation décimale d'un entier dans la gamme de –32 768 à 32 767, cette chaîne peut alors être codée en précisant qu'on utilise ce tableau de vocabulaire, en donnant l'indice du tableau de vocabulaire, puis en codant l'entier comme entier algébrique de deux octets. Les nombres et matrices à virgule flottante, de tels nombres sont pris en charge de la même façon.

Pour assurer un traitement rapide sans sacrifier la concision, de nombreux composants d'un document Fast Infoset (comme les chaînes de caractères et les composants représentant des éléments d'information de l'infoset XML) sont calés à l'octet, alors que d'autres composants (comme les longueurs et les indices de tableaux de vocabulaire) ne sont pas nécessairement calés à l'octet mais se terminent toujours sur le dernier bit d'un octet. La notation de contrôle de codage ASN.1 (définie dans la Rec. UIT-T X.692 | ISO/CEI 8825-3) est utilisée pour fournir une spécification formelle de ces codages optimisés (voir le § A.2), mais l'utilisation d'outils ECN pour les implémentations n'est pas nécessaire et une description complète du codage est donnée à l'Annexe C.

Les tableaux de vocabulaire pour un document Fast Infoset particulier peuvent être initialisés par des informations en tête du document, et y sont habituellement ajoutés de façon dynamique, pour donner de la souplesse au codage. Les tableaux de vocabulaire initiaux peuvent être fournis par une référence à l'ensemble de tableaux de vocabulaire finaux d'autres documents Fast Infoset identifiés (ou par d'autres moyens). Cette référence de vocabulaire peut alors être complétée par d'autres ajouts de tableaux pour fournir les tableaux de vocabulaire initiaux pour ce document. D'autres ajouts dynamiques ultérieurs sont normalement faits aux tableaux durant la création ou le traitement du document.

Finalement, il est fourni un mécanisme de génération d'un document Fast Infoset qui inclut des données (appelées données de traitement supplémentaire) qui se rapportent au traitement supplémentaire facultatif du document Fast Infoset, lié à un URI qui identifie une spécification complète de la forme et de la sémantique des données de traitement supplémentaire. Les données de traitement supplémentaire facultatif sont ignorées par les traitements ultérieurs du document Fast Infoset si l'URI n'est pas connu, ou si le traitement qu'elles spécifient n'est pas accepté ou pas utile.

NOTE – Un exemple de telles données de traitement supplémentaire serait celui de données fournissant des indices permettant un accès immédiat à des parties du document Fast Infoset, de sorte que le document tout entier n'ait pas besoin d'être traité si son seul intérêt réside dans les parties du document Fast Infoset qui correspondent à une étiquette XML spécifique.

L'Annexe A fait partie intégrante de la présente Recommandation | Norme internationale, et contient un module ASN.1 (voir la Rec. UIT-T X.680 | ISO/CEI 8824-1) et deux modules ECN (EDM et ELM – voir la Rec. UIT-T X.692 | ISO/CEI 8825-3) qui spécifient conjointement le contenu abstrait et le codage de niveau binaire d'une valeur du type **`Document`**, qui porte la valeur d'une instance de l'ensemble d'informations XML du W3C.

L'Annexe B fait partie intégrante de la présente Recommandation | Norme internationale, et contient la spécification d'un type de support MIME qui identifie un document Fast Infoset.

L'Annexe C ne fait pas partie intégrante de la présente Recommandation | Norme internationale, et donne une description complète des codages spécifiés formellement au § 12 et au § A.2.

L'Annexe D ne fait pas partie intégrante de la présente Recommandation | Norme internationale, et donne des exemples de documents Fast Infoset générés à partir de documents XML. L'annexe D donne aussi la taille de la représentation XML et de la représentation Fast Infoset de ces exemples.

NORME INTERNATIONALE
RECOMMANDATION UIT-T

Technologies de l'information – Applications génériques de l'ASN.1: Fast Infoset

1 Domaine d'application

La présente Recommandation | Norme internationale spécifie un type d'ASN.1 (voir la Rec. UIT-T X.680 | ISO/CEI 8824-1) dont les valeurs abstraites représentent des instances de l'ensemble d'informations XML du W3C. Elle spécifie aussi les codages binaires pour ces valeurs, en utilisant la notation de contrôle de codage ASN.1 (voir la Rec. UIT-T X.692 | ISO/CEI 8825-3).

NOTE – Ces codages sont appelés documents Fast Infoset.

La présente Recommandation | Norme internationale spécifie aussi des techniques qui:

- minimisent la taille des documents Fast Infoset;
- maximisent la vitesse de création et de traitement des documents Fast Infoset;
- permettent la spécification (grâce au générateur d'un document Fast Infoset) de données de traitement supplémentaire.

Les deux premières techniques impliquent l'utilisation de tableaux de vocabulaire conceptuel. L'ensemble des tableaux de vocabulaire et la nature de leurs entrées est défini de façon complète dans la présente Recommandation | Norme internationale, mais leur représentation dans les mémoires d'ordinateurs est en dehors du domaine d'application de la présente Recommandation | Norme internationale. Les dispositions pour le transfert ou le stockage, ou la notation formelle de l'affichage ou la spécification des tableaux de vocabulaire à utiliser comme vocabulaire externe sont également en dehors du domaine d'application de la présente Recommandation | Norme internationale.

La troisième technique implique la fourniture de données de traitement supplémentaire et d'un URI qui identifie la forme et la sémantique de ces données. La spécification de formes particulières de données de traitement supplémentaire et leur utilisation sont en dehors du domaine d'application de la présente Recommandation | Norme internationale.

Les URI peuvent être utilisés pour identifier les vocabulaires finaux qui peuvent être utilisés comme tout ou partie d'un nouveau vocabulaire initial, mais l'affectation d'URI spécifiques à des vocabulaires finaux particuliers est en dehors du domaine d'application de la présente Recommandation | Norme internationale.

La présente Recommandation | Norme internationale spécifie des alphabets restreints prédéfinis, l'ajout par énumération à des tableaux de vocabulaire d'autres alphabets restreints, et l'utilisation de ces tableaux de vocabulaire pour un codage efficace des chaînes de caractères.

La présente Recommandation | Norme internationale spécifie aussi des algorithmes de codage prédéfinis pour le codage optimal de certaines chaînes de caractères et l'ajout à des tableaux de vocabulaire d'autres algorithmes de codage identifiés par des URI, mais la définition de ces autres algorithmes de codage et de leurs URI associés est en dehors du domaine d'application de la présente Recommandation | Norme internationale.

De plus, la présente Recommandation | Norme internationale spécifie un type de support d'extensions de messagerie Internet multiobjets (MIME, *multipurpose Internet mail extensions*) qui identifie un document Fast Infoset.

2 Références normatives

Les Recommandations | Normes internationales et autres références suivantes contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour la présente Recommandation | Norme internationale. Au moment de la publication, les éditions indiquées étaient en vigueur. Toutes Recommandations, Normes internationales et autres références sont sujettes à révision et les parties prenantes aux accords fondés sur la présente Recommandation | Norme internationale sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des Recommandations | Normes internationales et autres références indiquées ci-après. Le Bureau de la normalisation des télécommunications de l'UIT tient à jour une liste des Recommandations de l'UIT-T en vigueur. Les membres de la CEI et de l'ISO possèdent le registre des Normes internationales en vigueur. L'IETF tient à jour une liste des RFC, ainsi que de celles qui ont été rendues obsolètes par des RFC plus récents. Le W3C tient à jour la liste des

spécifications du W3C en vigueur. La référence à un document figurant dans la présente Recommandation | Norme internationale ne donne pas à ce document, en tant que tel, le statut d'une Recommandation ou Norme internationale.

2.1 Recommandations | Normes internationales identiques

- Recommandation UIT-T X.667 (2004) | ISO/CEI 9834-8:2005, *Technologies de l'information – Interconnexion des systèmes ouverts – Procédures opérationnelles des organismes d'enregistrement de l'OSI: génération et enregistrement des identificateurs uniques universels (UUID) et utilisation de ces identificateurs comme composants d'identificateurs d'objet ASN.1*.
- Recommandation UIT-T X.680 (2002) | ISO/CEI 8824-1:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification de la notation de base*.
- Recommandation UIT-T X.681 (2002) | ISO/CEI 8824-2:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des objets informationnels*. †
- Recommandation UIT-T X.682 (2002) | ISO/CEI 8824-3:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification des contraintes*. †
- Recommandation UIT-T X.683 (2002) | ISO/CEI 8824-4:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: paramétrage des spécifications de la notation de syntaxe abstraite numéro un*. †
- Recommandation UIT-T X.690 (2002) | ISO/CEI 8825-1:2002, *Technologies de l'information – Règles de codage ASN.1: spécification des règles de codage de base, des règles de codage canoniques et des règles de codage distinctives*. †
- Recommandation UIT-T X.691 (2002) | ISO/CEI 8825-2:2002, *Technologies de l'information – Règles de codage ASN.1: spécification des règles de codage compact (PER)*. †
- Recommandation UIT-T X.692 (2002) | ISO/CEI 8825-3:2002, *Technologies de l'information – Règles de codage ASN.1: notation de contrôle de codage (ECN)*.
- Recommandation UIT-T X.693 (2001) | ISO/CEI 8825-4:2002, *Technologies de l'information – Règles de codage ASN.1: règles de codage XML (XER)*. †

NOTE – L'ensemble complet des Recommandations | Normes internationales sur l'ASN.1 est énuméré ci-dessus, car toutes peuvent être applicables dans des utilisations particulières de la présente Recommandation | Norme internationale. Lorsqu'elles ne sont pas directement référencées dans le corps de la présente Recommandation | Norme internationale, un symbole † est ajouté à la référence.

2.2 Références supplémentaires

- ISO 8601:2004, *Eléments de données et formats d'échange – Echange d'informations – Représentation de la date et de l'heure*.
- ISO/CEI 10646:2003, *Technologies de l'information – Ensemble universel de caractères codés sur plusieurs octets*.
- Unicode, *The Unicode Standard, Version 4.0, The Unicode Consortium. (Reading, MA, Addison-Wesley).* (Norme Unicode, version 4.0)

  NOTE 1 – Les caractères graphiques (et leurs codages) définis par Unicode sont identiques à ceux définis par la norme ISO/CEI 10646-1, mais Unicode est inclus comme référence parce qu'il spécifie aussi les noms des caractères de contrôle et définit l'abréviation UTF-16BE.
- W3C XML 1.0:2004, *Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut national de recherche en informatique et en automatique, Keio University), http://www.w3.org/TR/2000/REC-xml-20040204/*. (Langage de balisage extensible, troisième édition)
- W3C XML 1.1:2004, *Extensible Markup Language (XML) 1.1, W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut national de recherche en informatique et en automatique, Keio University), http://www.w3.org/TR/2000/REC-xml11-20040204/*. (Langage de balisage extensible)

  NOTE 2 – Les références à W3C XML 1.0 et W3C XML 1.1 sont toutes deux incluses car aucune n'est un sous-ensemble de l'autre. Ces références ne sont utilisées qu'au § 3.4.10.
- W3C XML Information Set:2004, *XML Information Set (Second Edition), W3C Recommendation, Copyright © [04 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut national de recherche en informatique et en automatique, Keio University), http://www.w3.org/TR/2004/REC-xml-infoset-20040204/*. (Ensemble d'informations XML, deuxième édition)

– W3C XML Namespaces 1.0:1999, *Namespaces in XML, W3C Recommendation, Copyright © [14 January 1999] World Wide Web Consortium (Massachusetts Institute of Technology, Institut national de recherche en informatique et en automatique, Keio University), http://www.w3.org/TR/1999/REC-xm-lnames-19990114/*. (Espaces de nom en XML)

– W3C XML Namespaces 1.1:2004, *Namespaces in XML 1.1, W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut national de recherche en informatique et en automatique, Keio University), http://www.w3.org/TR/2004/REC-xm-l-names11-20040204/*. (Espaces de nom en XML)

NOTE 3 – Les références à W3C XML Namespaces 1.0 et W3C XML Namespaces 1.1 sont toutes deux incluses car aucune n'est un sous-ensemble de l'autre. Ces références ne sont utilisées qu'au § 3.4.10.

– IETF RFC 2045 (1996), *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. (Extensions de messagerie Internet multiobjets, MIME)*

– IETF RFC 2396 (1998), *Uniform Resource Identifiers (URI): Generic Syntax. (Identificateurs universels de ressources (URI): Syntaxe générique)*

– IEEE 754 (1985), *IEEE Standard for Binary Floating-Point Arithmetic. (Norme IEEE pour arithmétique binaire à virgule flottante)*

3 Définitions

Pour les besoins de la présente Recommandation | Norme internationale, on applique les définitions suivantes.

3.1 Termes ASN.1

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec UIT-T X.680 | ISO/CEI 8824-1:

a) type choix (*choice type*);

b) type séquence (*sequence type*);

c) type séquence de (*sequence-of type*).

3.2 Termes ECN

La présente Recommandation | Norme internationale utilise les termes suivants définis dans la Rec. UIT-T X.692 | ISO/CEI 8825-3:

a) module de définition de codage (EDM, *encoding definition module*);

b) module de lien de codage (ELM, *encoding link module*).

3.3 Termes de l'ISO/CEI 10646

La présente Recommandation | Norme internationale utilise les termes suivants définis dans l'ISO/CEI 10646:

a) plan multilingue de base.

3.4 Définitions supplémentaires

3.4.1 Base64: mécanisme de codage qui représente une valeur de chaîne d'octet comme une chaîne de caractères en utilisant un alphabet restreint de 65 caractères (voir le § 10.3 et la RFC 2045 de l'IETF).

3.4.2 chaîne de caractères: chaîne de caractères abstraits ISO/CEI 10646, sans aucune implication sur la façon dont ils sont codés.

3.4.3 algorithme de codage: spécification précise de la façon de coder efficacement une chaîne de caractères avec des caractéristiques spécifiées dans les octets.

NOTE – Un exemple serait le codage d'une chaîne telle que "-32176" en entier binaire complément à deux sur deux octets. Le codage sur deux octets serait accompagné d'un indice de tableau de vocabulaire identifiant cet algorithme de codage.

3.4.4 vocabulaire externe: ensemble de tableaux de vocabulaire référencé par un URI (voir au § 7.2.14).

3.4.5 document Fast Infoset: Infoset XML représenté comme spécifié dans la présente Recommandation | Norme internationale.

3.4.6 vocabulaire final: contenu des tableaux de vocabulaire à la fin de la création ou du traitement d'un document Fast Infoset.

3.4.7 élément informationnel: chacun des types d'éléments qui constitue un infoset XML.

3.4.8 vocabulaire initial: ensemble des tableaux de vocabulaire établis par information au début d'un document Fast Infoset qui fait facultativement référence à un vocabulaire externe et fournit facultativement des entrées de tableau supplémentaires.

3.4.9 suppléant de nom: ensemble de trois indices de tableau de vocabulaire (les deux premiers sont facultatifs) qui sont utilisés pour représenter un nom qualifié (voir au § 3.4.11).

3.4.10 document XML bien conformé à l'espace des noms: document XML W3C 1.0 bien conformé selon l'espace de noms XML 1.0 de la W3C ou document XML 1.1 W3C bien conformé selon l'espace de noms XML 1.1 de la W3C.

3.4.11 nom qualifié: ensemble comportant les propriétés **[prefix]**, **[namespace name]**, et **[local name]** d'un élément informationnel **element** ou d'un élément informationnel **attribute**.

3.4.12 alphabet restreint: ensemble ordonné de caractères ISO/CEI 10646 distincts, qui permet un codage compact de toute chaîne de caractères uniquement constituée de caractères pris dans cet ensemble.

3.4.13 indice de tableau de vocabulaire: valeur entière positive identifiant une entrée dans un tableau de vocabulaire.

3.4.14 tableaux de vocabulaire: ensemble de tableaux de concepts (normalement, mais pas nécessairement, construits de façon dynamique) associés à un document Fast Infoset, qui contient des chaînes de caractères ou d'autres informations, et prend en charge l'utilisation de valeurs entières positives normalement petites (indices de tableau de vocabulaire) qui identifient leurs entrées.

NOTE – Des exemples de tableaux de vocabulaire sont ceux qui contiennent des chaînes de caractères qui sont les propriétés **[local name]** des éléments d'information **attribute** ou **element**, ou des chaînes de caractère correspondant à des séquences d'éléments d'information **character** qui sont membres de la propriété **[children]** des éléments d'information **element**.

3.4.15 déclaration XML: codage UTF-8 d'une chaîne de caractère spécifiée (voir aussi le § 12.3) qui peut être incluse au début d'un document Fast Infoset pour identifier le codage comme celui d'un document Fast Infoset et pour le distinguer d'un document W3C XML 1.0 ou W3C XML 1.1.

3.4.16 infoset XML: ensemble de données abstraites qui décrit les informations dans un document XML à espace de nom bien conformé, comme spécifié dans l'ensemble d'informations XML W3C.

3.4.17 espace blanc XML: un ou plusieurs caractères TABULATION HORIZONTALE (9), SAUT DE LIGNE (10), RETOUR CHARIOT (13), ou ESPACE (32) d'Unicode.

NOTE – Ces caractères sont ceux qui correspondent à la production de "S" à la fois dans W3C XML 1.0 et W3C XML 1.1 (voir W3C XML 1.0, 2.3 et W3C XML 1.1, 2.3). Les caractères LIGNE SUIVANTE (133) et SEPARATEUR DE LIGNE (8232), qui peuvent survenir dans un document W3C XML 1.1 à espaces de nom bien conformés (voir W3C XML 1.1, 2.11), sont convertis en caractères SAUT DE LIGNE par un traitement de fin de ligne (voir W3C XML 1.1, 2.11). Si ces caractères surviennent dans un infoset XML généré à partir d'un document W3C XML 1.1 à espaces de nom bien conformés, ce ne sont pas des espaces blancs XML.

4 Abréviations

Pour les besoins de la présente Recommandation | Norme internationale, on applique les abréviations suivantes:

ASN.1	Notation de syntaxe abstraite numéro un (*abstract syntax notation one*)
BMP	Table multilingue de base (*basic multilingual plane*)
ECN	Notation de contrôle de codage (*encoding control notation*)
MIME	Extensions de messagerie Internet multiobjets (*multipurpose Internet mail extensions*)
UBL	Langage d'affaires universel (*universal business language*)
URI	Identificateur universel de ressources (*uniform resource identifier*)
UTF-8	Format de transformation UCS 8 (*universal transformation function 8-bit*) (voir l'ISO/CEI 10646, Annexe D)
UTF-16BE	Format de transformation UCS 16 bits gros-boutistes (*universal transformation function 16-bit Big Endian*) (voir Unicode, 2.6)
UUID	Identificateur unique universel (*universally unique identifier*)

XML Langage de balisage extensible (*extended markup language*)

5 Notation

5.1 La présente Recommandation | Norme internationale utilise la notation ASN.1 définie par la Rec. UIT-T X.680 | ISO/CEI 8824-1 pour la définition formelle des types de données dont les codages sont des documents Fast Infoset.

NOTE – Le paragraphe 12 spécifie l'application de la Rec. UIT-T X.692 | ISO/CEI 8825-3 aux définitions des types ASN.1, et donne le codage au niveau binaire d'un document Fast Infoset.

5.2 Dans la présente Recommandation | Norme internationale, la police de caractère **`Courier gras`** est utilisée pour les notations en ASN.1 et la police de caractères **Arial gras** est utilisée pour la syntaxe XML W3C et pour les noms des éléments d'information de l'ensemble d'informations XML.

5.3 Les noms des propriétés des éléments d'information sont en **Arial gras** et inclus entre crochets (par exemple, **[children]**).

5.4 Les noms des catégories de chaînes de caractères (voir le § 8.4.2) et les noms des catégories de noms qualifiés (voir le § 8.5.4) sont en MAJUSCULES.

5.5 Dans la présente Recommandation | Norme internationale, les positions des bits au sein d'un octet sont spécifiées en utilisant la terminologie "premier bit, second bit, etc., jusqu'au huitième bit", où le premier bit est le bit de plus fort poids de l'octet, et le huitième bit est le bit de plus faible poids de l'octet.

6 Principes de construction et d'utilisation des tableaux de vocabulaire

6.1 Les tableaux de vocabulaire sont des tables de concepts qui transposent un indice de tableau de vocabulaire en entrée de tableau de vocabulaire.

NOTE – La représentation des tableaux de vocabulaire dans les mémoires des ordinateurs n'est pas définie, pas plus que les moyens par lesquels une implémentation transpose un indice de tableau de vocabulaire en entrée de tableau de vocabulaire pour ce tableau.

6.2 Le créateur d'un document Fast Infoset à partir d'un infoset XML détermine le contenu des tableaux de vocabulaire.

6.3 Dans le cas le plus général, l'en-tête d'un document Fast Infoset peut faire référence à un ensemble de tableaux de vocabulaire (un vocabulaire externe), suivi par la spécification d'ajouts à ces tableaux de vocabulaire pour former le vocabulaire initial de ce document Fast Infoset. Des ajouts ultérieurs à ces tableaux de vocabulaire surviennent lors de la création et durant le traitement d'un document Fast Infoset, de sorte qu'ils croissent progressivement pour former les tableaux de vocabulaire finaux pour ce document.

6.4 Certains tableaux de vocabulaire croissent progressivement à partir d'un vocabulaire initial jusqu'à un vocabulaire final durant la création et pendant le traitement d'un document Fast Infoset, et ont donc le mot "dynamique" dans le nom du tableau de vocabulaire. Il n'existe pas de mécanisme de suppression des entrées d'un tableau.

6.5 Les indices d'un tableau de vocabulaire sont alloués de façon implicite. La première entrée de tout tableau de vocabulaire a un indice de tableau de vocabulaire de un, et chaque entrée suivante de ce tableau a la valeur de l'entier immédiatement supérieur d'indice de tableau de vocabulaire. Lorsque la présente Recommandation | Norme internationale spécifie que quelque chose doit être ajouté à un tableau de vocabulaire, cela implique qu'on doit lui allouer le prochain indice de tableau de vocabulaire disponible.

NOTE – Les indices de tableau de vocabulaire débutent à un et non à zéro parce que la valeur zéro (lorsqu'elle est permise) a la signification particulière de "chaîne de caractères vide" dans un champ qui pourrait autrement contenir un indice de tableau de vocabulaire.

6.6 Pour prendre en charge l'allocation implicite des indices de tableau de vocabulaire, l'ordre conceptuel de traitement des composants (de toute nature) d'un document Fast Infoset est entièrement défini (voir le § 8.1).

NOTE – Cet ordre est le même que l'ordre de codage des composants dans un document Fast Infoset. Cela n'implique pas nécessairement que la sémantique portée par le document soit traitée dans cet ordre. L'ordre n'est défini que pour garantir que le même indice de tableau de vocabulaire est alloué pour toute entrée de tableau de vocabulaire donnée à la fois par le créateur et par celui qui va traiter le document Fast Infoset.

6.7 Les tableaux de vocabulaire servent à de nombreux usages (voir le paragraphe 8), mais leur fonction première est de permettre l'utilisation d'un indice de tableau de vocabulaire à la place d'une entrée de tableau de vocabulaire, car de tels indices sont plus petits (et peuvent être traités plus vite) que les entrées de tableau. Un certain nombre d'entrées prédéfinies pour des tableaux de vocabulaire sont spécifiées au § 9. Ces entrées sont toujours implicitement présentes dans ces tableaux de vocabulaire, avec les indices de tableau de vocabulaire spécifiés au § 9.

6.8 Pour certaines catégories de chaînes de caractères, le créateur d'un document Fast Infoset a la faculté d'ajouter ou non une chaîne à un tableau de vocabulaire, selon le nombre attendu (ou connu) d'occurrences de cette chaîne de caractères dans l'infoset XML.

6.9 La forme et la signification précises des entrées d'un tableau de vocabulaire est spécifiée au § 8, mais ce sont dans la plupart des cas des chaînes de caractères de longueur variable, souvent courtes, mais pouvant aller jusqu'à 2^{32} octets.

6.10 Un créateur de document Fast Infoset conforme est obligé de faire tous les ajouts aux tableaux de vocabulaire spécifiés aux § 7.13.7, 7.14.6 à 7.14.7 et 7.16.7. Ceci garantit que le nombre d'entrées de tableau de vocabulaire dans chaque tableau de vocabulaire ne dépasse jamais 2^{20}.

NOTE – Une entrée de tableau de vocabulaire peut être égale à une ou plusieurs autres entrées de tableau de vocabulaire. Cela permet la création efficace de documents Fast Infoset. Toutefois, des entrées en double diminueront l'efficacité du transfert. Un processeur n'est pas affecté par des entrées en double.

6.11 Un traitement de document Fast Infoset conforme est nécessaire pour effectuer tous les ajouts aux tableaux de vocabulaire spécifiés aux § 7.13.8, 7.14.11 et 7.16.8. Ceci garantit que la restriction du § 6.10 a n'a pas été violée.

7 Définitions de type ASN.1

7.1 Généralités

7.1.1 La présente Recommandation | Norme internationale spécifie un ensemble de types ASN.1 qui prennent en charge une représentation de l'ensemble d'informations XML. Le type racine de cet ensemble de types est le type **`Document`**.

7.1.2 Quelques restrictions sont imposées sur le contenu des infosets XML et on a fait quelques simplifications dans la représentation (voir le § 11) afin d'améliorer la facilité d'utilisation de la spécification et l'efficacité des codages produits avec son aide.

NOTE – Un infoset XML qui ne satisfait pas à ces restrictions ne peut pas être présenté comme document Fast Infoset, et ne peut non plus être représenté normalement comme un document XML à espace de noms bien conformé.

7.1.3 Pour chaque sorte d'élément d'information spécifié dans l'ensemble d'informations XML W3C, une définition de type ASN.1 correspondante est fournie dans la présente Recommandation | Norme internationale. Cette définition de type est toujours un type de séquence, avec des composants correspondant aux propriétés de l'élément d'information.

7.1.4 Certaines propriétés des éléments d'information ne sont pas incluses dans les définitions de type ASN.1 (voir le § 11.4).

7.1.5 Dans certains cas, la valeur d'une propriété qui n'est pas incluse dans les définitions de type ASN.1 peut être déterminée à partir de la valeur d'autres propriétés du même élément d'information ou d'autres éléments d'information inclus. Dans ces cas, l'omission de cette propriété simplifie la représentation sans perte d'information. Il y a cependant, quelques cas dans lesquels la valeur d'une propriété qui n'est pas incluse ne peut être déduite des autres propriétés. Dans de tels cas, l'omission de cette propriété est une simplification qui ne limite pas l'utilité de la spécification pour la plupart des cas d'utilisation pratiques.

7.1.6 La section 12 spécifie le codage du type **`Document`**.

7.2 Le type `Document`

7.2.1 Le type **`Document`** est:

```
Document ::= SEQUENCE {
      additional-data          SEQUENCE (TAILLE(1..one-meg)) DE
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } FACULTATIF,
      initial-vocabulary       SEQUENCE {
            external-vocabulary         URI FACULTATIF,
            restricted-alphabets        SEQUENCE (TAILLE(1..256)) DE
                  NonEmptyOctetString FACULTATIF,
            encoding-algorithms         SEQUENCE (TAILLE(1..256)) DE
                  NonEmptyOctetString FACULTATIF,
            prefixes                    SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            namespace-names             SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
```

```
            local-names                SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            other-ncnames              SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            other-uris                 SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            attribute-values           SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            content-character-chunks   SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            other-strings              SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            element-name-surrogates    SEQUENCE (TAILLE(1..one-meg)) DE
                  NameSurrogate FACULTATIF,
            attribute-name-surrogates  SEQUENCE (TAILLE(1..one-meg)) DE
                  NameSurrogate FACULTATIF }
                  (CONSTRAINED BY {
                        -- Si le composant initial-vocabulary est présent, au moins
                        -- un de ses composants doit être présent -- }) FACULTATIF,

      notations                  SEQUENCE (TAILLE(1..MAX)) DE
                                       Notation FACULTATIF,
      unparsed-entities          SEQUENCE (TAILLE(1..MAX)) DE
                                       UnparsedEntity FACULTATIF,
      character-encoding-scheme  NonEmptyOctetString FACULTATIF,
      standalone                 BOOLEAN FACULTATIF,
      version                    NonIdentifyingStringOrIndex FACULTATIF
                                   -- catégorie AUTRE CHAINE --,
      children                   SEQUENCE (TAILLE(0..MAX)) DE
            CHOIX {
                  element                    Element,
                  processing-instruction     ProcessingInstruction,
                  comment                    Comment,
                  document-type-declaration DocumentTypeDeclaration }}
```

où la valeur **one-meg** est:

```
one-meg ENTIER ::= 1048576 -- Deux à la puissance 20
```

Le type **NonEmptyOctetString** est:

```
NonEmptyOctetString ::= OCTET STRING (TAILLE(1..four-gig))
```

où la valeur **four-gig** est:

```
four-gig ENTIER ::= 4294967296 -- Deux à la puissance 32
```

Le type **URI** est:

```
URI ::= NonEmptyOctetString
```

7.2.2 Les types **EncodedCharacterString**, **NameSurrogate**, **Notation**, **UnparsedEntity**, **NonIdentifyingStringOrIndex**, **Element**, **ProcessingInstruction**, **Comment** et **DocumentTypeDeclaration** sont définis respectivement aux § 7.17, 7.15, 7.11, 7.10, 7.14, 7.3, 7.5, 7.8 et 7.9.

7.2.3 Le type **URI** doit être un URI tel que spécifié dans la RFC 2396 de l'IETF.

7.2.4 Le composant **restricted-alphabets** de **initial-vocabulary** (s'il est présent) doit comporter une ou plusieurs chaînes de caractères, chacune portant les caractères d'un alphabet restreint. Chaque chaîne de caractères doit contenir au moins deux caractères, et tous les caractères de la chaîne de caractères doivent être distincts.

NOTE – L'utilisation d'alphabets restreints pour optimiser le codage des chaînes de caractères est spécifiée au § 7.17.6.

7.2.5 Le composant **encoding-algorithms** de **initial-vocabulary** (s'il est présent) doit comporter un ou plusieurs URI dont chacun identifie un algorithme de codage.

NOTE – La présente Recommandation | Norme internationale définit des algorithmes de codage prédéfinis (voir le § 10), avec des indices de tableau de vocabulaire spécifiés, mais il est en dehors du domaine d'application de la présente Recommandation | Norme internationale de définir plus avant les algorithmes de codage et les URI qui leur sont associés, ni les moyens de définir de tels algorithmes. Les informations nécessaires pour définir un algorithme de codage sont spécifiées au § 8.3.3.

7.2.6 Le type **Document** représente l'élément d'information **document** d'un infoset XML. Comme tous les autres éléments d'information d'un infoset XML sont des propriétés de cet élément d'information ou des propriétés d'un

élément fils ou descendant de cet élément (à un degré quelconque), chaque **Document** représente un infoset XML complet.

NOTE – Chaque **Document** dépourvu de référence à un vocabulaire externe (voir le § 7.2.13) définit aussi un vocabulaire final qui peut être utilisé comme vocabulaire externe de certains autres documents Fast Infoset.

7.2.7 Le composant **additional-data** (s'il est présent) doit comporter un ou plusieurs composants **additional-datum** pour permettre des mécanismes supplémentaires de traitement d'un document Fast Infoset.

NOTE 1 – Par exemple des données permettant à un traitement d'accéder à des parties d'un document Fast Infoset sans avoir à traiter la totalité du document. La forme de telles données n'est pas normalisée.

NOTE 2 – Le nombre des composants **additional-datum** est limité à 2^{20} composants (voir le § 7.2.1).

7.2.8 Chaque **additional-datum** doit comporter:

a) Le composant **id** (valeur du type **URI**); l'URI doit faire référence à une spécification qui définit la forme et la sémantique du composant **data**.

NOTE – La forme de **additional-datum** peut être spécifiée comme type abstrait conjointement avec une règle de codage, ou par tout autre moyen convenable.

b) Le composant **data**, qui est une chaîne d'octets qui contient les données de traitement supplémentaire.

7.2.9 L'utilisation d'un composant **additional-data** est soumise aux conditions suivantes:

a) un composant **additional-datum** peut être ignoré par un traitement sauf si l'URI est reconnu et si le traitement supplémentaire est considéré comme pertinent pour l'activité de ce traitement;

b) un traitement qui ignore tous les composants **additional-datum** est à tout le moins capable de générer un infoset XML équivalent à l'infoset XML utilisé pour générer le document Fast Infoset.

7.2.10 Des composants **additional-datum** multiples avec le même URI peuvent être présents, et seront traités conformément à la spécification associée à l'URI.

7.2.11 Le composant **initial-vocabulary** fournit des données qui (conjointement avec des entrées de tableau prédéfinies) déterminent complètement le contenu initial du tableau d'alphabet restreint (voir le § 8.2), le tableau d'algorithme de codage (voir le § 8.3), les tableaux de chaînes dynamiques (voir le § 8.4), et les tableaux de noms dynamiques (voir le § 8.5) de ce document Fast Infoset (le vocabulaire initial du document Fast Infoset). Un vocabulaire initial comporte les données suivantes:

a) un ensemble ordonné d'alphabets restreints (voir le § 8.2.2), contenant au moins les alphabets restreints prédéfinis (voir le § 9);

b) un ensemble ordonné d'algorithmes de codage (voir le § 8.3.2), contenant au moins les algorithmes de codage prédéfinis (voir le § 10);

c) huit ensembles ordonnés indépendants de chaînes de caractères, correspondant aux huit catégories de chaînes de caractères spécifiées dans la présente Recommandation | Norme internationale (voir le § 8.4.2), chaque ensemble contenant zéro ou plus chaînes de caractères d'une seule catégorie;

d) deux ensembles ordonnés indépendants de substituts de nom (voir le § 8.5.2), correspondant aux deux catégories de noms qualifiés spécifiées dans la présente Recommandation | Norme internationale (voir le § 8.5.4), chaque ensemble contenant zéro ou plus substituts de nom d'une seule catégorie.

NOTE – Un vocabulaire initial ne peut être totalement vide, parce qu'il contient toujours (au moins) les alphabets restreints prédéfinis et les algorithmes de codage prédéfinis. Cependant, il ne serait pas anormal pour un document Fast Infoset d'avoir un vocabulaire initial ne contenant que ces données, car la décision (du créateur d'un document Fast Infoset) sur la manière d'utiliser le composant **initial-vocabulary** dépend de l'implémentation, et certaines implémentations peuvent choisir d'ajouter toutes les entrées de tableau de vocabulaire de façon dynamique (dans le corps du document Fast Infoset).

7.2.12 Le vocabulaire initial du document Fast Infoset doit être déterminé comme suit:

a) Si le composant **initial-vocabulary** est absent, le vocabulaire initial doit alors uniquement comporter les entrées de tableau prédéfinies spécifiées aux § 7.2.21, 7.2.22, et aux § 9 et 10.

b) Si le composant **initial-vocabulary** est présent, et que le composant **external-vocabulary** est absent, le vocabulaire initial doit alors comporter les entrées de tableau prédéfinies spécifiées aux § 7.2.21, 7.2.22, et aux § 9 et 10, avec les éventuelles entrées de tableau ajoutées déterminées par le § 7.2.16.

c) Si le composant **initial-vocabulary** est présent, et que le composant **external-vocabulary** est présent, le vocabulaire initial doit alors comporter le vocabulaire final identifié par le composant **external-vocabulary** comme spécifié aux § 7.2.13 et 7.2.14, avec les éventuelles entrées de tableau de vocabulaire prédéfinies déterminées par le § 7.2.16.

7.2.13 Le composant **`external-vocabulary`** identifie un vocabulaire final qui utilise un des mécanismes spécifiés au § 7.2.14. Le type d'URI (voir le § 7.2.1) détermine le vocabulaire final à utiliser comme vocabulaire externe selon une des trois méthodes (voir le § 7.2.14).

NOTE – La présente Recommandation | Norme internationale ne spécifie aucun vocabulaire externe ni aucun URI qui fait référence à des vocabulaires externes. De tels vocabulaires externes et URI peuvent être définis par une autorité capable d'allouer ces URI, et peuvent faire l'objet d'un accord privé ou peuvent être normalisés.

7.2.14 Un vocabulaire externe peut être spécifié selon l'une des trois méthodes suivantes:

a) comme vocabulaire final d'un document Fast Infoset, qui ne doit pas lui-même faire référence à un vocabulaire externe; ou

NOTE 1 – C'est une question d'implémentation de savoir si le vocabulaire final est mémorisé localement ou si seul le document Fast Infoset est mémorisé et si le vocabulaire final est généré durant le traitement.

NOTE 2 – La restriction que le vocabulaire final d'un document Fast Infoset avec référence à un vocabulaire externe ne peut être utilisé lui-même comme vocabulaire externe est imposée dans le but de simplifier l'implémentation et éviter les références en boucle.

b) comme document XML à espace de nom bien conformé, qui est traité conceptuellement comme suit:

1) l'infoset XML du document XML à espace de nom bien conformé doit être déterminé;

2) le document Fast Infoset pour cet infoset XML doit être créé comme spécifié dans la présente Recommandation | Norme internationale, mais il ne doit pas avoir de composant **`initial-vocabulary`**, le composant **`add-to-table`** du **`NonIdentifyingStringOrIndex`** (voir le § 7.14) doit toujours être mis à **`VRAI`** et plusieurs chaînes de caractères identiques ne doivent pas être présentes dans un tableau de chaîne;

3) le vocabulaire final de ce document Fast Infoset devient le vocabulaire externe;

NOTE 3 – C'est une question d'implémentation de savoir si le vocabulaire final est mémorisé localement ou si seul le document Fast Infoset est mémorisé et si le vocabulaire final est généré durant le traitement.

c) comme un ensemble de tableaux de vocabulaire spécifiés en utilisant tout autre mécanisme ou texte suffisamment précis, qui doit inclure les entrées de tableau prédéfinies des § 9 et 10 (avec les indices de tableau de vocabulaire spécifiés par ces sections).

NOTE 4 – Il est en dehors du domaine d'application de la présente Recommandation | Norme internationale de spécifier une notation pour la définition des tableaux de vocabulaire.

NOTE 5 – L'exigence d'inclusion des entrées de tableau prédéfinies lors de l'utilisation de ce mécanisme sert à garantir que tous les tableaux de vocabulaire incluent les entrées de tableau prédéfinies.

7.2.15 Pour un vocabulaire externe spécifié conformément au § 7.2.14 c, toutes les chaînes et entrées de tableau de nom, excepté celles des tableaux PREFIX et NAMESPACE NAME, doivent se voir allouer des indices consécutifs débutant à 1. Les entrées des tableaux PREFIX et NAMESPACE NAME doivent recevoir des indices consécutifs débutant à 2. Tous les alphabets restreints autres que ceux qui sont prédéfinis doivent recevoir des indices consécutifs débutant à 16. Tous les algorithmes de codage autres que ceux qui sont prédéfinis doivent recevoir des indices consécutifs débutant à 32.

7.2.16 Chaque **`NonEmptyOctetString`**, **`EncodedCharacterString`** et **`NameSurrogate`** éventuellement présent dans tout composant restant de **`initial-vocabulary`** doit être ajouté dans l'ordre (voir le § 8.1) à un tableau de vocabulaire, comme spécifié au Tableau 1.

Tableau 1/X.891 – Transposition des identifiants de composant en tableaux de vocabulaire

Identifiant de composant	Type de l'entrée ASN.1	Tableau de vocabulaire (voir le § 8)
`restricted-alphabets`	`NonEmptyOctetString`	Tableau d'alphabet restreint (voir § 8.2)
`encoding-algorithms`	`NonEmptyOctetString`	Tableau d'algorithme de codage (voir § 8.3)
`prefixes`	`NonEmptyOctetString`	Tableau PREFIX (voir § 8.4)
`namespace-names`	`NonEmptyOctetString`	Tableau NAMESPACE NAME (voir § 8.4)
`local-names`	`NonEmptyOctetString`	Tableau LOCAL NAME (voir § 8.4)
`other-ncnames`	`NonEmptyOctetString`	Tableau OTHER NCNAME (voir § 8.4)
`other-uris`	`NonEmptyOctetString`	Tableau OTHER URI (voir § 8.4)

Tableau 1/X.891 – Transposition des identifiants de composant en tableaux de vocabulaire

Identifiant de composant	Type de l'entrée ASN.1	Tableau de vocabulaire (voir le § 8)
`attribute-values`	`EncodedCharacterString`	Tableau ATTRIBUTE VALUE (voir § 8.4)
`content-character-chunks`	`EncodedCharacterString`	Tableau CONTENT CHARACTER CHUNK (voir § 8.4)
`other-strings`	`EncodedCharacterString`	Tableau OTHER STRING (voir § 8.4)
`element-name-surrogates`	`NameSurrogate`	Tableau ELEMENT NAME (voir § 8.5)
`Attribute-name-surrogates`	`NameSurrogate`	Tableau ATTRIBUTE NAME (voir § 8.5)

7.2.17 Une valeur du type **`NonEmptyOctetString`** doit porter le codage UTF-8 (voir l'ISO/CEI 10646, Annexe D) d'une chaîne de caractères.

7.2.18 Dans un vocabulaire initial, le tableau d'alphabet restreint et le tableau d'algorithme de codage doivent avoir au plus 256 entrées. Tous les autres tableaux doivent avoir au plus 2^{20} entrées.

NOTE – La restriction sur le nombre d'entrées est destinée à assurer des limites supérieures communes aux indices des tableaux. La restriction s'applique aussi si les entrées de tableau sont ajoutées de façon dynamique (voir les § 7.13.7, 7.14.6, 7.14.7 et 7.16.7). Ces restrictions n'empêchent le codage d'aucun infoset XML comme document Fast Infoset.

7.2.19 Les alphabets restreints prédéfinis ont des indices de tableau de vocabulaire compris entre 1 et 2 (voir le § 9). Les indices de tableau de vocabulaire des alphabets restreints dans le composant **`restricted-alphabets`** de **`initial-vocabulary`** (s'il est présent) doivent être alloués comme suit:

a) s'il n'y a pas de vocabulaire externe, ou si le vocabulaire externe contient seulement des alphabets restreints prédéfinis, les indices doivent alors être alloués en commençant à 16;

b) autrement, les indices doivent être alloués en commençant à un de plus que l'indice d'alphabet restreint le plus élevé du vocabulaire externe.

NOTE – Ceci signifie que les indices de tableau de vocabulaire de 3 à 15 ne sont pas utilisés. Ces valeurs sont réservées pour de futures versions de la présente Recommandation | Norme internationale.

7.2.20 Les algorithmes de codage prédéfinis ont des indices de tableau de vocabulaire compris entre 1 et 10 (voir le § 10). Les indices de tableau de vocabulaire des algorithmes de codage dans le composant **`encoding-algorithms`** de **`initial-vocabulary`** (s'il est présent) doivent être alloués comme suit:

a) s'il n'y a pas de vocabulaire externe, ou si le vocabulaire externe ne contient que des algorithmes de codage prédéfinis, les indices doivent alors être alloués à partir de 32;

b) autrement, les indices doivent être alloués en partant de un de plus que le plus haut indice d'algorithme de codage dans le vocabulaire externe.

NOTE – Ceci signifie que les indices de tableau de vocabulaire de 11 à 31 ne sont pas utilisés. Ces valeurs sont réservées pour de futures versions de la présente Recommandation | Norme internationale.

7.2.21 Le tableau PREFIX doit avoir une entrée de préfixe prédéfinie "xml", à laquelle est alloué un indice de 1. Les indices de tableau de vocabulaire des préfixes dans le composant **`prefixes`** de **`initial-vocabulary`** (s'il est présent) doivent être alloués comme suit:

a) s'il n'y a pas de vocabulaire externe, ou si le vocabulaire externe ne contient que l'entrée de préfixe prédéfinie, les indices doivent alors être alloués à partir de 2;

b) autrement, les indices doivent être alloués en partant de un plus l'indice le plus haut de préfixe dans le vocabulaire externe.

7.2.22 Le tableau NAMESPACE NAME doit avoir une entrée de nom d'espace de nom prédéfinie de:

http://www.w3.org/XML/1998/namespace

Il doit lui être alloué un indice de 1.

7.2.23 Les indices de tableau de vocabulaire des noms d'espace de nom dans le composant **`namespace-names`** de **`initial-vocabulary`** (s'il est présent) doivent être alloués comme suit:

a) s'il n'y a pas de vocabulaire externe, ou si le vocabulaire externe ne contient que l'entrée de nom d'espace de nom prédéfinie, les indices doivent alors être alloués en partant de 2;

b) autrement, les indices doivent être alloués en partant de un plus l'indice de nom d'espace de nom le plus élevé dans le vocabulaire externe.

7.2.24 Le composant **notations** représente la propriété **[notations]** de l'élément d'information **document.** Le type de ce composant est un type sequence-of, même si la propriété **[notations]** est spécifiée dans l'ensemble d'informations XML W3C comme ensemble non ordonné (d'éléments d'information **notation**).

NOTE – Ici et ailleurs, un type sequence-of est utilisé plutôt qu'un type set-of parce que ce dernier ne satisfait pas à l'exigence des ordonnancement strict de tous les composants d'un document Fast Infoset (voir le § 8.1).

7.2.25 Le composant **unparsed-entities** représente la propriété **[unparsed entities]** de l'élément d'information **document.** Le type de ce composant est un type sequence-of, même si la propriété **[unparsed entities]** est spécifiée dans un ensemble d'informations XML W3C comme un ensemble non ordonné (d'éléments d'information **unparsed entity**).

7.2.26 Le composant **character-encoding-scheme** représente la propriété **[character encoding scheme]** de l'élément d'information **document.** Le type de ce composant est **NonEmptyOctetString** et une valeur de ce composant doit porter le codage UTF-8 (voir l'ISO/CEI 10646, Annexe D) de la propriété **[character encoding scheme].** L'absence de ce composant dans une valeur abstraite du type **Document** indique que la propriété **[character encoding scheme]** a une valeur de "UTF-8".

NOTE – La prise en charge de la propriété **[character encoding scheme]** permet l'aller-retour entre des documents XML et des documents Fast Infoset sans changement de schéma de codage de caractères. Un créateur de document Fast Infoset à partir d'un document XML peut coder la propriété **[character encoding scheme]** obtenue à partir de la déclaration de codage du document XML (voir W3C XML 1.0, 4.3.1 et W3C XML 1.1, 4.3.1). Le traitement d'un document Fast Infoset peut utiliser le composant **character-encoding-scheme** (s'il est présent) s'il souhaite produire le codage d'origine.

7.2.27 Le component **standalone** représente la propriété **[standalone]** de l'élément d'information **document.** La valeur abstraite **TRUE** (*vrai*) représente la valeur **yes** (*oui*) de cette propriété et la valeur abstraite **FALSE** (*faux*) représente la valeur **no** (*non*). L'absence de ce composant dans une valeur abstraite du type **Document** indique que la propriété **[standalone]** n'a pas de valeur.

7.2.28 Le composant **version** représente la propriété **[version]** de l'élément d'information **document**. Le type de ce composant est **NonIdentifyingStringOrIndex** (voir le § 7.14), qui représente ici une chaîne de caractères de la catégorie OTHER STRING. L'absence de ce composant dans une valeur abstraite du type **Document** indique que la propriété **[version]** n'a pas de valeur.

7.2.29 Le composant **children** (*fils*) représente la propriété **[children]** de l'élément d'information **document**. Exactement un des éléments de la sequence-of (à toute position) doit utiliser l'**element** de remplacement du type de choix, et au moins un des éléments (à toute position) doit utiliser la **document-type-declaration** de remplacement. Chacun des autres éléments éventuels doit utiliser la **processing-instruction** de remplacement ou le **comment** de remplacement.

7.2.30 La propriété **[document element]** de l'élément d'information **document** n'est pas incluse dans le type **Document**. La valeur de cette propriété est toujours le seul et unique élément d'information **element** qui soit un membre de la propriété **[children]** de l'élément d'information **document.**

7.2.31 La propriété **[base URI]** de l'élément d'information **document** n'est pas incluse dans le type **Document** et n'est pas prise en charge dans la présente Recommandation | Norme internationale.

7.2.32 La propriété **[all declarations processed]** de l'élément d'information **document** n'est pas incluse dans le type **Document**, et est supposée avoir la valeur **true** (voir le § 11.3).

7.3 Le type Element

7.3.1 Le type **Element** est:

```
Element ::= SEQUENCE {
      namespace-attributes    SEQUENCE (TAILLE(1..MAX)) DE
            NamespaceAttribute FACULTATIF,
      qualified-name          QualifiedNameOrIndex
                                    -- catégorie ELEMENT NAME --,
      attributes              SEQUENCE (TAILLE(1..MAX)) DE
            Attribute OPTIONAL,
      children                SEQUENCE (TAILLE(0..MAX)) DE
            CHOICE {
                  element                       Element,
                  processing-instruction        ProcessingInstruction,
                  unexpanded-entity-reference   UnexpandedEntityReference,
                  character-chunk               CharacterChunk,
                  comment                       Comment }}
```

7.3.2 Les types **NameSpaceAttribute**, **QualifiedNameOrIndex**, **Attribute**, **ProcessingInstruction**, **UnexpandedEntityReference**, **CharacterChunk** et **Comment** sont définis respectivement aux § 7.12, 7.16, 7.4, 7.5, 7.6, 7.7 et 7.8.

7.3.3 Le type **Element** représente l'élément d'information **element** de l'ensemble d'informations XML.

7.3.4 Le composant **namespace-attributes** représente la propriété **[namespace attributes]** de l'élément d'information **element**. Le type de ce composant est un type sequence-of, même si la propriété **[namespace attributes]** est spécifiée dans l'ensemble d'informations XML W3C comme un ensemble non ordonné (des éléments d'information **attribute**).

NOTE – Le type du composant de la sequence-of est **NamespaceAttribute** (plutôt que **Attribute**), même si la propriété **[namespace attributes]** de l'élément d'information **element** est spécifiée dans l'ensemble d'informations XML W3C comme un ensemble des éléments d'information **attribute**. Dans un infoset XML restreint (voir le § 11.3), les propriétés d'un élément d'information **namespace** peuvent être déterminées à partir des propriétés d'un élément d'information **attribute** qui représente un attribut d'espace de nom. L'inverse n'est que partiellement vrai, mais cette limitation est considérée comme acceptable pour les utilisations attendues de la présente Recommandation | Norme internationale. (Voir aussi la note du § 7.2.24.)

7.3.5 Le composant **qualified-name** représente le nom qualifié (voir le § 3.4.11) de l'élément d'information **element** (qui est l'ensemble comportant les propriétés **[prefix]**, **[namespace name]** et **[local name]** de cet élément d'information). Le type de ce composant est **QualifiedNameOrIndex** (voir le § 7.16), qui représente ici un nom qualifié de la catégorie ELEMENT NAME.

7.3.6 Le composant **attributes** représente la propriété **[attributes]** de l'élément d'information **element**. Le type de ce composant est un type sequence-of, même si la propriété **[attributes]** est spécifiée dans l'ensemble d'informations XML W3C comme un ensemble non ordonné (d'éléments d'information **attribute**).

7.3.7 Le composant **children** représente la propriété **[children]** de l'élément d'information **element**. Lorsque deux ou plus fils adjacents sont des éléments d'information **character**, un seul élément **CharacterChunk** peut être utilisé pour représenter ces éléments d'information **character** adjacents.

NOTE – S'il y a une séquence de N caractères adjacents parmi les fils d'un élément d'information **element**, tout regroupement de ces N caractères en une série de tronçons de caractères consécutifs est permis. Cependant, on s'attend à ce que le créateur d'un document Fast Infoset fasse chaque tronçon de caractères aussi grand que possible afin de produire des codages efficaces.

7.3.8 La propriété **[in-scope namespaces]** de l'élément d'information **element** n'est pas incluse dans le type **Element**.

NOTE – Dans un infoset XML restreint (voir le § 11.3), la propriété **[in-scope namespaces]** d'un élément d'information **element** peut être déterminée à partir de la propriété **[namespace attributes]** de l'élément d'information **element**, conjointement avec la propriété **[namespace attributes]** éventuellement de tous les éléments d'information **element** qui contiennent (directement ou indirectement) cet élément d'information **element**.

7.3.9 La propriété **[base URI]** de l'élément d'information **element** n'est pas incluse dans le type **Element** et n'est pas prise en charge dans la présente Recommandation | Norme internationale.

7.3.10 La propriété **[parent]** de l'élément d'information **element** n'est pas incluse dans le type **Element**. La valeur de cette propriété, pour tout élément d'information **element** donné, est l'élément d'information **document** ou **element** qui contient cet élément d'information comme membre de sa propriété **[children]**.

7.4 Le type Attribute

7.4.1 Le type **Attribute** est:

```
Attribute ::= SEQUENCE {
      qualified-name      QualifiedNameOrIndex
                                -- catégorie ATTRIBUTE NAME --,
      normalized-value    NonIdentifyingStringOrIndex
                                -- catégorie ATTRIBUTE VALUE -- }
```

7.4.2 Les types **QualifiedNameOrIndex** et **NonIdentifyingStringOrIndex** sont définis respectivement aux § 7.16 et 7.14.

7.4.3 Le type **Attribute** représente l'élément d'information **attribute** de l'ensemble d'informations XML.

7.4.4 Le composant **qualified-name** représente le nom qualifié (voir le § 3.4.11) de l'élément d'information **attribute** (qui est l'ensemble comportant les propriétés **[prefix]**, **[namespace name]** et **[local name]** de cet élément d'information). Le type de ce composant est **QualifiedNameOrIndex** (voir le § 7.16), qui représente ici un nom qualifié de la catégorie ATTRIBUTE NAME.

7.4.5 Le composant **normalized-value** représente la propriété **[normalized value]** de l'élément d'information **attribute**. Le type de ce composant est **NonIdentifyingStringOrIndex** (voir le § 7.14), qui représente ici une chaîne de caractères de la catégorie ATTRIBUTE VALUE.

7.4.6 La longueur de la chaîne de caractère allouée à **`normalized-value`** ne peut pas être supérieure à 2^{32}.

NOTE – Cette restriction est impliquée par la définition de l'ASN.1, qui est conçu pour optimiser les codages et pour simplifier l'implémentation (voir aussi le § 11.3 j).

7.4.7 La propriété **[specified]** de l'élément d'information **attribute** n'est pas incluse dans le type **`Attribute`**.

7.4.8 La propriété **[attribute type]** de l'élément d'information **attribute** n'est pas incluse dans le type **`Attribute`**.

7.4.9 La propriété **[references]** de l'élément d'information **attribute** n'est pas incluse dans le type **`Attribute`**.

NOTE – Dans un infoset XML restreint (voir le § 11.3), la propriété **[references]** d'un élément d'information **attribute** peut être déterminée à partir de la propriété **[normalized value]** de l'élément d'information **attribute**, conjointement avec les propriétés des autres éléments d'information de l'infoset XML.

7.4.10 La propriété **[owner element]** de l'élément d'information **attribute** n'est pas incluse dans le type **`Attribute`**. La valeur de cette propriété, pour tout élément d'information **attribute** donné, est l'élément d'information **element** qui contient cet élément d'information comme membre de sa propriété **[attributes]**.

7.5 Le type `ProcessingInstruction`

7.5.1 Le type **`ProcessingInstruction`** est:

```
ProcessingInstruction ::= SEQUENCE {
      target       IdentifyingStringOrIndex
                           -- catégorie OTHER NCNAME --,
      content      NonIdentifyingStringOrIndex
                           -- catégorie OTHER STRING -- }
```

7.5.2 Les types **`IdentifyingStringOrIndex`** et **`NonIdentifyingStringOrIndex`** sont respectivement définis aux § 7.13 et 7.14.

7.5.3 Le type **`ProcessingInstruction`** représente l'élément d'information **processing instruction** de l'élément d'informations XML.

7.5.4 Le composant **`target`** représente la propriété **[target]** de l'élément d'information **processing instruction.** Le type de ce composant est **`IdentifyingStringOrIndex`** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER NCNAME.

7.5.5 Le composant **`content`** représente la propriété **[content]** de l'élément d'information **processing instruction**. Le type de ce composant est **`NonIdentifyingStringOrIndex`** (voir le § 7.14), qui représente ici une chaîne de caractères de la catégorie OTHER STRING.

7.5.6 La longueur de la chaîne de caractères allouée à **`content`** ne peut pas être supérieure à 2^{32}.

NOTE – Cette restriction est impliquée par la définition de l'ASN.1, qui est conçu pour l'optimisation des codages et la simplification des implémentations (voir aussi le § 11.3 j).

7.5.7 La propriété **[notation]** de l'élément d'information **processing instruction** n'est pas incluse dans le type **`ProcessingInstruction`**.

NOTE – Dans un infoset XML restreint (voir le § 11.3), la propriété **[notation]** d'un élément d'information **processing instruction** peut être déterminée à partir de la propriété **[target]** de l'élément d'information **processing instruction** conjointement avec la propriété **[notations]** de l'élément d'information **document**.

7.5.8 La propriété **[parent]** de l'élément d'information **processing instruction** n'est pas incluse dans le type **`ProcessingInstruction`**. La valeur de cette propriété pour tout élément d'information **processing instruction** donné, est l'élément d'information **document**, **element** ou **document type definition** qui contient cet élément d'information comme membre de sa propriété **[children]**.

7.6 Le type `UnexpandedEntityReference`

7.6.1 Le type **`UnexpandedEntityReference`** est:

```
UnexpandedEntityReference ::= SEQUENCE {
      name                 IdentifyingStringOrIndex
                                 -- catégorie OTHER NCNAME --,
      system-identifier    IdentifyingStringOrIndex FACULTATIF
                                 -- catégorie OTHER URI --,
      public-identifier    IdentifyingStringOrIndex FACULTATIF
                                 -- catégorie OTHER URI -- }
```

7.6.2 Le type **`IdentifyingStringOrIndex`** est défini au § 7.13.

7.6.3 Le type **UnexpandedEntityReference** représente l'élément d'information **unexpanded entity reference** l'ensemble d'information XML.

7.6.4 Le composant **name** représente la propriété **[name]** de l'élément d'information **unexpanded entity reference**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER NCNAME.

7.6.5 Le composant **system-identifier** représente la propriété **[system identifier]** de l'élément d'information **unexpanded entity reference**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER URI. L'absence de ce composant dans une valeur abstraite du type **UnexpandedEntityReference** indique que la propriété **[system identifier]** n'a pas de valeur.

7.6.6 Le composant **public-identifier** représente la propriété **[public identifier]** de l'élément d'information **unexpanded entity reference**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER URI. L'absence de ce composant dans une valeur abstraite du type **UnexpandedEntityReference** indique que la propriété **[public identifier]** n'a pas de valeur.

7.6.7 La propriété **[declaration base URI]** de l'élément d'information **unexpanded entity reference** n'est pas incluse dans le type **UnexpandedEntityReference** et n'est pas prise en charge dans la présente Recommandation | Norme internationale.

7.6.8 La propriété **[parent]** de l'élément d'information **unexpanded entity reference** n'est pas incluse dans le type **UnexpandedEntityReference**. La valeur de cette propriété, pour tout élément d'information **unexpanded entity reference** donné, est l'élément d'information **element** qui contient cet élément d'information comme membre de sa propriété **[children]**.

7.7 Le type CharacterChunk

7.7.1 Le type **CharacterChunk** est:

```
CharacterChunk ::= SEQUENCE {
      character-codes             NonIdentifyingStringOrIndex
                                        -- catégorie CONTENT CHARACTER CHUNK -- }
```

7.7.2 Le type **NonIdentifyingStringOrIndex** est défini au § 7.14.

7.7.3 Le type **CharacterChunk** correspond à l'élément d'information **character**, mais représente une série d'éléments d'information **character** adjacents (membres du **[children]** de l'élément d'information parent **element**) plutôt qu'un seul élément d'information **character**.

7.7.4 Le nombre d'éléments d'information **character** représenté par une valeur du type **CharacterChunk** ne doit pas être zéro.

7.7.5 Le composant **character-codes** représente la propriété **[character code]** de ou des éléments d'information **character** dans le tronçon. Le type de ce composant est **NonIdentifyingStringOrIndex** (voir le § 7.14), qui représente ici une chaîne de caractères de la catégorie CONTENT CHARACTER CHUNK.

7.7.6 La longueur de la chaîne de caractères allouée à **character-codes** ne peut être supérieure à 2^{32}.

NOTE – Cette restriction est impliquée par la définition de l'ASN.1, qui est conçu pour l'optimisation des codages et la simplification des implémentations. Elle n'empêche pas le codage d'un élément d'information **element** qui contient plus de 2^{32} éléments d'information **character**, parce que plusieurs tronçons peuvent être utilisés.

7.7.7 La propriété **[element content whitespace]** du ou des éléments d'information **character** n'est pas incluse dans le type **CharacterChunk**.

7.7.8 La propriété **[parent]** du ou des éléments d'information **character** n'est pas incluse dans le type **CharacterChunk**. La valeur de cette propriété, pour tout élément d'information **character** donné, est l'élément d'information **element** qui contient cet élément d'information comme membre de sa propriété **[children]**.

7.8 Le type Comment

7.8.1 Le type **Comment** est:

```
Comment ::= SEQUENCE {
      content     NonIdentifyingStringOrIndex -- catégorie OTHER STRING --}
```

7.8.2 Le type **NonIdentifyingStringOrIndex** est défini au § 7.14.

7.8.3 Le type **Comment** représente l'élément d'information **comment** de l'ensemble d'information XML.

7.8.4 Le composant **content** représente la propriété **[content]** de l'élément d'information **comment**. Le type de ce composant est **NonIdentifyingStringOrIndex** (voir le § 7.14), qui représente ici une chaîne de caractères de la catégorie OTHER STRING.

7.8.5 La longueur de la chaîne de caractères allouée à **content** ne peut pas être supérieure à 2^{32}.

NOTE – Cette restriction est impliquée par la définition de l'ASN.1, qui est conçu pour l'optimisation des codages et la simplification des implémentations (voir aussi le § 11.3 j).

7.8.6 La propriété **[parent]** de l'élément d'information **comment** n'est pas incluse dans le type **Comment**. La valeur de cette propriété, pour tout élément d'information **comment** donné, est l'élément d'information **document** ou **element** qui contient cet élément d'information comme membre de sa propriété **[children]**.

7.9 Le type DocumentTypeDeclaration

7.9.1 Le type **DocumentTypeDeclaration** est:

```
DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier    IdentifyingStringOrIndex FACULTATIF
                                 -- catégorie OTHER URI --,
      public-identifier    IdentifyingStringOrIndex FACULTATIF
                                 -- catégorie OTHER URI --,
      children             SEQUENCE (SIZE(0..MAX)) DE
            ProcessingInstruction }
```

7.9.2 Le type **IdentifyingStringOrIndex** est défini au § 7.13.

7.9.3 Le type **DocumentTypeDeclaration** représente l'élément d'information **document type declaration** de l'ensemble d'informations XML.

7.9.4 Le composant **system-identifier** représente la propriété **[system identifier]** de l'élément d'information **document type declaration**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER URI. L'absence de ce composant dans une valeur abstraite du type **DocumentTypeDeclaration** indique que la propriété **[system identifier]** n'a pas de valeur.

7.9.5 Le composant **public-identifier** représente la propriété **[public identifier]** de l'élément d'information **document type declaration**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER URI. L'absence de ce composant dans une valeur abstraite du type **DocumentTypeDeclaration** indique que la propriété **[public identifier]** n'a pas de valeur.

7.9.6 Le composant **children** représente la propriété **[children]** de l'élément d'information **document type declaration**.

7.9.7 La propriété **[parent]** de l'élément d'information **document type declaration** n'est pas incluse dans le type **DocumentTypeDeclaration**. La valeur de cette propriété, pour tout élément d'information **document type declaration** donné, est l'élément d'information **document** qui contient cet élément d'information comme membre de sa propriété **[children]**.

7.10 Le type UnparsedEntity

7.10.1 Le type **UnparsedEntity** est:

```
UnparsedEntity ::= SEQUENCE {
      name                 IdentifyingStringOrIndex
                                 -- catégorie OTHER NCNAME --,
      system-identifier    IdentifyingStringOrIndex
                                 -- catégorie OTHER URI --,
      public-identifier    IdentifyingStringOrIndex FACULTATIF
                                 -- catégorie OTHER URI --,
      notation-name        IdentifyingStringOrIndex
                                 -- catégorie OTHER NCNAME -- }
```

7.10.2 Le type **IdentifyingStringOrIndex** est défini au § 7.13.

7.10.3 Le type **UnparsedEntity** (*entité non analysée*) représente l'élément d'information **unparsed entity** de l'ensemble d'informations XML.

7.10.4 Le composant **name** représente la propriété **[name]** de l'élément d'information **unparsed entity**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER NCNAME.

7.10.5 Le composant **system-identifier** représente la propriété **[system identifier]** de l'élément d'information **unparsed entity**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER URI.

7.10.6 Le composant **public-identifier** représente la propriété **[public identifier]** de l'élément d'information **unparsed entity**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER URI. L'absence de ce composant dans une valeur abstraite du type **UnparsedEntity** indique que la propriété **[public identifier]** n'a pas de valeur.

7.10.7 Le composant **notation-name** représente la propriété **[notation name]** de l'élément d'information **unparsed entity**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER NCNAME.

7.10.8 La propriété **[declaration base URI]** de l'élément d'information **unparsed entity** n'est pas incluse dans le type **UnparsedEntity** et n'est pas pris en charge par la présente Recommandation | Norme internationale.

7.10.9 La propriété **[notation]** de l'élément d'information **unparsed entity** n'est pas incluse dans le type **UnparsedEntity**.

NOTE – Dans un infoset XML restreint (voir le § 11.3), la propriété **[notation]** d'un élément d'information **unparsed entity** peut être déterminée à partir de la propriété **[notation name]** de l'élément d'information **unparsed entity** conjointement avec la propriété **[notations]** de l'élément d'information **document**.

7.11 Le type Notation

7.11.1 Le type **Notation** est:

```
Notation ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                -- catégorie OTHER NCNAME --,
      system-identifier     IdentifyingStringOrIndex FACULTATIF
                                -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                -- catégorie OTHER URI -- }
```

7.11.2 Le type **IdentifyingStringOrIndex** est défini au § 7.13.

7.11.3 Le type **Notation** représente l'élément d'information **notation** de l'ensemble d'informations XML.

7.11.4 Le composant **name** représente la propriété **[name]** de l'élément d'information **notation**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER NCNAME.

7.11.5 Le composant **system-identifier** représente la propriété **[system identifier]** de l'élément d'information **notation**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER URI. L'absence de ce composant dans une valeur abstraite du type **Notation** indique que la propriété **[system identifier]** n'a pas de valeur.

7.11.6 Le composant **public-identifier** représente la propriété **[public identifier]** de l'élément d'information **notation**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie OTHER URI. L'absence de ce composant dans une valeur abstraite du type **Notation** indique que la propriété **[public identifier]** n'a pas de valeur.

7.11.7 La propriété **[declaration base URI]** de l'élément d'information **notation** n'est pas incluse dans le type **Notation** et n'est pas prise en charge dans la présente Recommandation | Norme internationale.

7.12 Le type NamespaceAttribute

7.12.1 Le type **NamespaceAttribute** est:

```
NamespaceAttribute ::= SEQUENCE {
      prefix              IdentifyingStringOrIndex FACULTATIF
                                -- catégorie PREFIX --,
      namespace-name      IdentifyingStringOrIndex  FACULTATIF
                                -- catégorie NAMESPACE NAME -- }
```

7.12.2 Le type **IdentifyingStringOrIndex** est défini au § 7.13.

7.12.3 Le type **NamespaceAttribute** représente un élément d'information **attribute** qui est un membre de la propriété **[namespace attributes]** d'un élément d'information **element** de l'ensemble d'informations XML.

NOTE – Dans l'ensemble d'informations XML, les attributs d'attribut et d'espace de nom sont des éléments d'information **attribute**. Dans la présente Recommandation | Norme internationale, différents types sont utilisés aux fins d'optimisation.

7.12.4 Il y a deux types d'attributs d'espace de nom dans l'ensemble d'informations XML:

a) des déclarations d'espace de nom par défaut: la propriété **[prefix]** de l'élément d'information **attribute** n'a pas de valeur, et la propriété **[local name]** est **"xmlns"**;

b) des déclarations d'espace de nom pas par défaut: la propriété **[prefix]** de l'élément d'information **attribute** est **"xmlns"**, et la propriété **[local name]** donne le préfixe de la déclaration d'espace de nom.

Dans les deux cas, la propriété **[normalized value]** de l'élément d'information **attribute** donne le nom d'espace de nom de la déclaration d'espace de nom.

7.12.5 Si l'attribut d'espace de nom est une déclaration d'espace de nom par défaut (cas a) du § 7.12.4), le composant **prefix** doit alors être absent, autrement (cas b du § 7.12.4) il doit être présent, représentant la propriété **[local name]** de l'élément d'information **attribute**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie PREFIX.

7.12.6 Si la propriété **[normalized value]** de l'élément d'information **attribute** est une chaîne vide, le composant **namespace-name** doit alors être absent, autrement il doit être présent, représentant la propriété **[normalized value]** de l'élément d'information **attribute**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie NAMESPACE NAME.

7.12.7 La propriété **[namespace name]** de l'élément d'information **attribute** est toujours **"http://www.w3.org/2000/xmlns/"** (voir Infoset XML W3C) et n'est pas incluse dans le type **NamespaceAttribute**.

7.13 Le type IdentifyingStringOrIndex

7.13.1 Le type **IdentifyingStringOrIndex** est:

```
IdentifyingStringOrIndex ::= CHOICE {
      literal-character-string   NonEmptyOctetString,
      string-index               INTEGER (1..one-meg) }
```

7.13.2 Le type **NonEmptyOctetString** et la valeur **one-meg** sont définis au § 7.2.1.

7.13.3 Le type **IdentifyingStringOrIndex** représente une chaîne de caractères qui porte les informations d'identification.

NOTE – Des exemples de telles chaînes de caractères sont les préfixes, noms d'espace de nom, et noms locaux des éléments et attributs.

7.13.4 Une valeur abstraite de ce type ASN.1 contient soit une chaîne de caractères (d'une catégorie donnée) comme valeur du type **NonEmptyOctetString**, soit l'indice de tableau de vocabulaire d'une chaîne de caractères (d'une catégorie donnée) dans le tableau de vocabulaire pour cette catégorie de chaîne (voir le § 8.4.2), appelée le tableau de chaîne applicable.

NOTE 1 – La catégorie de chaîne est toujours spécifiée dans le texte associé des paragraphes précédents (voir les § 7.5 à 7.12) chaque fois que ce type est utilisé.

NOTE 2 – Les chaînes de caractères "Identifiantes" sont traitées différemment des chaînes de caractères "non identifiantes" (voir le § 7.14). Alors qu'une chaîne de caractères non identifiante peut être codée dans un choix de nombreux formats de codage, toutes les chaînes de caractères identifiantes sont codées en UTF-8. Aussi, alors qu'une chaîne de caractères non identifiante peut être ou non (au choix de son créateur) ajoutée au tableau de la chaîne dynamique (voir le § 7.14.6), les chaînes de caractères identifiantes sont toujours ajoutées au tableau de chaîne dynamique (voir le § 7.13.7).

7.13.5 La chaîne **literal-character-string**, si elle est présente, doit porter le codage UTF-8 (voir la norme ISO/CEI 10646, Annexe D) de la chaîne de caractères (voir le § 7.13.4).

7.13.6 L'indice **string-index**, s'il est présent, doit contenir l'indice de tableau de vocabulaire de l'une quelconque des entrées du tableau de chaîne applicable qui sont identiques à la chaîne de caractères.

7.13.7 Lors de la création d'une valeur abstraite de ce type ASN.1 représentant une chaîne de caractères donnée (d'une catégorie donnée), s'il existe une chaîne de caractères identique dans le contenu en cours du tableau de chaîne applicable, le créateur doit effectuer l'action a) ou l'action b) ci-dessous à titre d'option d'implémentation (mais la première option devrait être sélectionnée, si possible, car elle produit le plus petit nombre d'indices pointés vers la même chaîne de caractères), autrement (il n'existe aucune chaîne de caractères identique), le créateur doit effectuer l'action b) ci-dessous. Les actions a) et b) consistent en ce qui suit:

a) choisir l'indice **string-index** de remplacement, et allouer au **string-index** l'indice de tableau de vocabulaire de l'une quelconque des entrées existantes qui sont identiques à la chaîne de caractères;

b) choisir la chaîne **literal-character-string** de remplacement, allouer la chaîne de caractères donnée à **literal-character-string** et ajouter au tableau de chaîne applicable une chaîne de caractères identique, à moins que ce tableau ne contienne déjà 2^{20} entrées.

NOTE – Le choix d'effectuer l'action b) engendrera plus d'une chaîne de caractères identique dans le tableau de chaîne (s'il ne contient pas déjà 2^{20} entrées). Cela n'affectera pas le traitement ultérieur des chaînes de caractères (voir le § 7.13.8).

7.13.8 Lors du traitement d'une valeur abstraite de ce type ASN.1 représentant une chaîne de caractères (d'une catégorie donnée), le traitement doit déterminer la chaîne de caractères représentée par la valeur abstraite comme suit:

a) si l'indice **string-index** de remplacement est présent, la chaîne de caractères représentée par la valeur abstraite doit alors être la chaîne de caractères dans le contenu en cours du tableau de chaîne applicable dont l'indice de tableau de vocabulaire est la valeur de **string-index**;

b) si la chaîne **literal-character-string** de remplacement est présente, la chaîne de caractères représentée par la valeur abstraite doit alors être la valeur de **literal-character-string**, et une chaîne de caractères identique doit être ajoutée au tableau de chaîne applicable (mais voir le § 7.13.9), à moins que ce tableau ne contienne déjà 2^{20} entrées.

7.13.9 Si un traitement est incapable (pour toute raison, y compris des limites spécifiques de l'implémentation) d'ajouter une chaîne à un tableau de vocabulaire contenant moins de 2^{20} entrées lorsqu'un tel ajout est requis par le § 7.13.8 b, il doit cesser le traitement du document Fast Infoset et doit produire une erreur.

7.14 Le type NonIdentifyingStringOrIndex

7.14.1 Le type **NonIdentifyingStringOrIndex** est:

```
NonIdentifyingStringOrIndex ::= CHOICE {
      literal-character-string    SEQUENCE {
            add-to-table                BOOLEAN,
            character-string            EncodedCharacterString },
      string-index                INTEGER (0..one-meg) }
```

7.14.2 Le type **EncodedCharacterString** et la valeur **one-meg** sont définis respectivement aux § 7.17 et 7.2.1.

7.14.3 Le type **NonIdentifyingStringOrIndex** représente une chaîne de caractères qui ne porte pas d'informations d'identification.

NOTE – Un exemple d'une telle chaîne de caractères est la valeur d'un attribut.

7.14.4 Une valeur abstraite du type **NonIdentifyingStringOrIndex** porte une chaîne de caractères (d'une catégorie donnée) comme valeur de type **EncodedCharacterString** (voir le § 7.17), ou l'indice de tableau de vocabulaire d'une chaîne de caractères d'une catégorie donnée dans le tableau de vocabulaire pour cette catégorie de chaîne (voir le § 8.4.2), appelé le tableau de chaîne applicable.

NOTE – La catégorie de chaîne est toujours spécifiée dans le texte associé des paragraphes antérieurs chaque fois que ce type est utilisé.

7.14.5 L'indice **string-index**, s'il est présent, doit être zéro (notant une chaîne de caractères de longueur zéro – voir le § 7.14.6) ou bien doit contenir l'indice de tableau de vocabulaire de l'une quelconque des entrées du tableau de chaîne applicable qui sont identiques à la chaîne de caractères.

7.14.6 Pour une chaîne de caractères de longueur zéro, le créateur doit toujours utiliser l'indice **string-index** de remplacement avec la valeur d'entier zéro. Un traitement doit considérer une telle valeur comme représentant une chaîne de caractères de longueur zéro.

7.14.7 Lors de la création d'une valeur abstraite du type **NonIdentifyingStringOrIndex** représentant une chaîne de caractères donnée (d'une catégorie donnée), de longueur différente de zéro, s'il existe une chaîne de caractères identique dans le contenu en cours du tableau de chaîne applicable, le créateur doit effectuer l'action a) ou l'action b) ci-dessous à titre d'option d'implémentation (mais la première option devrait être sélectionnée, si possible, car elle produit le plus petit nombre d'indices pointés vers la même chaîne de caractères), autrement (s'il n'existe aucune chaîne de caractères identique), le créateur doit effectuer l'action b) ci-dessous. Les actions a) et b) consistent en ce qui suit:

a) choisir l'indice **string-index** de remplacement et allouer à l'indice **string-index** l'indice de tableau de vocabulaire de l'une quelconque des entrées existantes qui sont identiques à la chaîne de caractères;

b) choisir la chaîne **literal-character-string** de remplacement, allouer la chaîne de caractères donnée au composant **character-string** et procéder de l'une des deux manières suivantes:

1) ajouter une chaîne de caractères identique au tableau de chaîne applicable et régler le composant **add-to-table** à **VRAI** (cette action b) 1) ne doit pas être utilisée si le tableau de chaîne applicable contient déjà 2^{20} entrées); ou

NOTE 1 – Si le tableau de chaîne applicable contient déjà 2^{20} entrées, seules l'action a) ou b) 2) peut être utilisé.

2) régler le composant **add-to-table** à **FAUX**

NOTE 2 – Le choix d'effectuer l'action b) 1) engendrera plus d'une chaîne de caractères identique dans le contenu en cours. Cela n'affectera pas le traitement ultérieur des chaînes de caractères (voir le § 7.14.8).

7.14.8 Lors du traitement de la valeur abstraite de ce type ASN.1 qui représente une chaîne de caractères d'une catégorie donnée, le traitement doit déterminer la chaîne de caractères représentée par la valeur abstraite comme suit:

a) si l'indice **string-index** de remplacement est présent, la chaîne de caractères représentée par la valeur abstraite doit alors être la chaîne de caractères dans le tableau de chaîne applicable dont l'indice de tableau de vocabulaire est la valeur de **string-index**.

NOTE 1 – Si le **string-index** dépasse la taille courante de ce tableau de vocabulaire, le document Fast Infoset est erroné.

b) si la chaîne **literal-character-string** de remplacement est présente et que le composant **add-to-table** a la valeur **VRAI**, la chaîne de caractères représentée par la valeur abstraite doit alors être la valeur du composant **character-string**. Le traitement doit ajouter au tableau de chaîne applicable une chaîne de caractères identique (mais voir le § 7.14.12).

NOTE 2 – Si le tableau de chaîne applicable contient déjà 2^{20} chaînes, le document Fast Infoset est erroné.

c) si la chaîne **literal-character-string** de remplacement est présente et que le composant **add-to-table** a la valeur **FAUX**, la chaîne de caractères représentée par la valeur abstraite doit alors être la valeur du composant **character-string**.

7.14.9 Si un traitement est incapable (pour toute raison, y compris tenant aux limites spécifiques de l'implémentation) d'ajouter une chaîne à un tableau de vocabulaire lorsqu'un tel ajout est requis par le § 7.14.8 b, il doit arrêter de traiter le document Fast Infoset et produire une erreur.

7.15 Le type NameSurrogate

7.15.1 Le type **NameSurrogate** est:

```
NameSurrogate ::= SEQUENCE {
      prefix-string-index           INTEGER(1..one-meg) OPTIONAL,
      namespace-name-string-index   INTEGER(1..one-meg) OPTIONAL,
      local-name-string-index       INTEGER(1..one-meg) }
      (CONSTRAINED BY  {-- prefix-string-index ne doit être présent que si
                       -- namespace-name-string-index est présent --})
```

7.15.2 La valeur **one-meg** est définie au § 7.2.1.

7.15.3 Le type **NameSurrogate** contient trois entiers positifs (dont les deux premiers sont facultatifs) qui forment un substitut de nom (voir le § 8.5.2).

NOTE – Ce type n'apparaît que dans le composant **initial-vocabulary** du type **Document**.

7.15.4 Les indices **prefix-string-index** (s'il est présent), **namespace-name-string-index** (s'il est présent) et **local-name-string-index** doivent être respectivement supérieurs à zéro et inférieurs au nombre d'entrées dans les tableaux PREFIX, NAMESPACE NAME, et LOCAL-NAME dans le vocabulaire initial.

NOTE – Si le traitement d'un document Fast Infoset choisit de ne pas vérifier si cette contrainte est satisfaite, il pourrait en résulter une faiblesse de la sécurité du traitement ultérieur.

7.15.5 L'indice **prefix-string-index** doit être absent sauf si **namespace-name-string-index** est présent.

7.16 Le type QualifiedNameOrIndex

7.16.1 Le type **QualifiedNameOrIndex** est:

```
QualifiedNameOrIndex ::= CHOICE {
      literal-qualified-name  SEQUENCE {
            prefix                IdentifyingStringOrIndex OPTIONAL
                                        -- catégorie PREFIX --,
            namespace-name        IdentifyingStringOrIndex OPTIONAL
                                        -- catégorie NAMESPACE NAME --,
            local-name            IdentifyingStringOrIndex
                                        -- catégorie LOCAL NAME -- },
      name-surrogate-index    INTEGER (1..one-meg) }
```

7.16.2 Le type **IdentifyingStringOrIndex** et la valeur **one-meg** sont définis respectivement aux § 7.13 et 7.2.1.

7.16.3 Le type **`QualifiedNameOrIndex`** représente un nom qualifié (voir le § 3.4.11).

7.16.4 Une valeur abstraite de ce type ASN.1 contient trois composants correspondant aux préfixe, nom d'espace de nom, et nom local d'un nom qualifié (d'une catégorie donnée), ou l'indice de tableau de vocabulaire d'un substitut de nom d'une catégorie donnée dans le tableau de vocabulaire pour cette catégorie de nom qualifié (voir le § 8.5.2), appelé le tableau de noms applicable.

NOTE – La catégorie est toujours spécifiée dans le texte associé des paragraphes précédents chaque fois que ce type est utilisé.

7.16.5 L'indice **`name-surrogate-index`** (*indice de substitut de nom*), s'il est présent, doit contenir l'indice de tableau de vocabulaire d'un substitut de nom dans le tableau de noms applicable.

7.16.6 Si **`namespace-name`** est absent, le **`prefix`** doit alors aussi être absent.

7.16.7 Lors de la création d'une valeur abstraite de ce type ASN.1 représentant un nom qualifié donné (d'une catégorie donnée), un créateur doit effectuer les actions spécifiées dans les paragraphes suivants.

7.16.7.1 Les conditions suivantes doivent être évaluées dans l'ordre:

a) soit le nom qualifié n'a pas de préfixe ou le préfixe du nom qualifié existe dans le contenu actuel du tableau PREFIX;

b) soit le nom qualifié n'a pas de nom d'espace de nom ou le nom d'espace de nome du nom qualifié existe dans le contenu actuel du tableau NAMESPACE NAME;

c) le nom local du nom qualifié existe dans le contenu actuel du tableau LOCAL NAME;

d) les trois premières conditions sont toutes satisfaites et un substitut de nom (voir le § 8.5), consistant en indices du tableau de vocabulaire des préfixe éventuellement, nom d'espace de nom éventuellement, et nom local, existent dans le contenu actuel du tableau de noms applicable.

7.16.7.2 Si toutes les conditions ci-dessus sont satisfaites, l'indice **`name-surrogate-index`** de remplacement doit être choisi, et doit être réglé à l'indice du tableau de vocabulaire du substitut de nom déterminé au § 7.16.7.1 d dans le tableau de noms applicable, complétant les procédures du § 7.16.7.

7.16.7.3 Autrement, le **`literal-qualified-name`** (*nom qualifié littéral*) de remplacement doit être choisi, et ses composants doivent être alloués comme suit:

a) si le nom qualifié n'a pas de préfixe, alors le composant **`prefix`** doit être absent, autrement le préfixe doit être alloué au composant **`prefix`** en appliquant le § 7.13.7, avec cette restriction que l'action 7.13.7 b) ne doit pas être exécutée s'il existe une chaîne de caractères identique dans le contenu en cours du tableau de chaîne applicable. Le type de ce composant est **`IdentifyingStringOrIndex`** (voir le § 7.13), qui représente ici une chaîne de caractères de catégorie PREFIX;

b) si le nom qualifié n'a pas de nom d'espace de nom, alors le composant **`namespace-name`** doit être absent, autrement le nom d'espace de nom doit être alloué au composant **`namespace-name`** en appliquant le § 7.13.7, avec cette restriction que l'action 7.13.7 b) ne doit pas être exécutée s'il existe une chaîne de caractères identique dans le contenu en cours du tableau de chaîne applicable. Le type de ce composant est **`IdentifyingStringOrIndex`** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie NAMESPACE NAME (voir le § 8.4.2);

c) le nom local du nom qualifié doit être alloué (en appliquant le § 7.13.7) au composant **`local-name`**. Le type de ce composant est **`IdentifyingStringOrIndex`** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie LOCAL NAME.

NOTE – L'application du § 7.13.7 à ce paragraphe peut provoquer l'ajout du nom local au tableau LOCAL NAME.

7.16.7.4 Si l'application du § 7.13.7 au § 7.16.7.3 d'une ou plusieurs des trois chaînes de caractères ci-dessus n'a pas ajouté la chaîne à un tableau de vocabulaire parce que le tableau contenait déjà 2^{20} entrées (voir le § 7.13.7 b), il ne peut alors être créé de substitut de nom pour ce nom qualifié.

7.16.7.5 Autrement, un substitut de nom (voir le § 8.5), comportant le ou les indices de tableau de vocabulaire du préfixe éventuel, de nom d'espace de nom éventuel, et de nom local, doit être créé. Si ce substitut de nom n'existe pas dans le contenu actuel du tableau de noms applicable, il doit alors être ajouté à ce tableau, à moins que le tableau ne contienne déjà 2^{20} entrées.

7.16.8 Lors du traitement d'une valeur abstraite du type **`QualifiedNameOrIndex`** représentant un nom qualifié d'une catégorie donnée, le traitement doit déterminer le nom qualifié représenté par la valeur abstraite conformément aux paragraphes suivants.

7.16.8.1 Si l'indice **name-surrogate-index** de remplacement est présent, le nom qualifié représenté par la valeur abstraite doit alors être celui représenté par le substitut de nom de la catégorie donnée (voir le § 8.5.2) dans le tableau de noms applicable dont l'indice de tableau de vocabulaire est la valeur de **name-surrogate-index**.

7.16.8.2 Si le nom **literal-qualified-name** de remplacement est présent, alors:

a) le nom qualifié représenté par la valeur abstraite doit être déterminé comme suit:

1) le préfixe du nom qualifié doit être déterminé (en appliquant le § 7.13.8) à partir du composant **prefix** (s'il est présent). Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie PREFIX;

2) le nom d'espace de nom du nom qualifié doit être déterminé (en appliquant le § 7.13.8) à partir du composant **namespace-name** (s'il est présent). Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie NAMESPACE NAME (voir le § 8.4.2);

3) le nom local du nom qualifié doit être déterminé (en appliquant le § 7.13.8) à partir du composant **local-name**. Le type de ce composant est **IdentifyingStringOrIndex** (voir le § 7.13), qui représente ici une chaîne de caractères de la catégorie LOCAL NAME.

NOTE – Ces actions peuvent exiger un ajout aux tableaux de vocabulaire correspondants, comme spécifié au § 7.13.8 b.

b) si, après de possibles ajouts résultant du traitement des composants de **literal-qualified-name**, des indices de tableau de vocabulaire sont disponibles pour tous les composants qui sont présents, un substitut de nom consistant en ces indices de vocabulaire doit alors être ajouté au tableau de noms applicable (mais voir le § 7.16.9), à moins que ce tableau de vocabulaire ne contienne déjà 2^{20} substituts de noms.

7.16.9 Si un traitement est incapable (pour toute raison y compris des limitations spécifiques de l'implémentation) d'ajouter un substitut de nom à un tableau de vocabulaire qui contient moins de 2^{20} entrées lorsqu'un tel ajout est requis par le § 7.16.8.2 b, il doit arrêter le traitement du document Fast Infoset et produire une erreur.

7.17 Le type EncodedCharacterString

7.17.1 Le type **EncodedCharacterString** est:

```
EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOICE {
            utf-8                   NULL,
            utf-16                  NULL,
            restricted-alphabet     INTEGER(1..256),
            encoding-algorithm      INTEGER(1..256) },
      octets                NonEmptyOctetString }
```

7.17.2 Le type **EncodedCharacterString** contient un codage d'une chaîne de caractères. Il spécifie une chaîne d'octets qui est une transposition réversible d'une chaîne de caractères en chaîne d'octets. Le créateur d'un document Fast Infoset spécifie dans **encoding-format** le codage utilisé, et le traitement utilise ces informations pour décoder les octets dans le composant **octets** en une chaîne de caractères.

7.17.3 Le composant **octets** doit porter une valeur de chaîne d'octets qui est un codage de la chaîne de caractères spécifié par le composant **encoding-format**.

NOTE – En général, il y a plusieurs codages qui peuvent être utilisés pour une chaîne de caractères donnée. Un créateur de document Fast Infoset peut choisir un codage fondé sur des critères clairs (par exemple, il peut essayer d'optimiser la taille ou la vitesse de traitement), mais il peut aussi préférer la commodité et l'applicabilité universelle de UTF-8 ou UTF-16BE. Il y a aussi le cas de certains codages qui ne seront capables de coder que des chaînes de caractères qui contiennent seulement un sous-ensemble des caractères ISO/CEI 10646.

7.17.4 Le format de codage **utf-8** peut être utilisé pour toute chaîne de caractères. Ce format de codage doit être appliqué en produisant un codage UTF-8 (voir l'ISO/CEI 10646) de la chaîne de caractères et en allouant ce codage au composant **octets**.

NOTE – Ce format de codage convient très bien pour les chaînes de caractères dans lesquelles les caractères ISO/CEI 10646 de la première partie du plan multilingue de base de l'ISO/CEI 10646 sont les plus communs, et où aucun des autres formats de codage n'est applicable ou plus utile.

7.17.5 Le format de codage **utf-16** peut aussi être utilisé pour toute chaîne de caractères. Ce format de codage doit être appliqué en produisant un codage UTF-16BE (voir Unicode, 2.6) de la chaîne de caractères et en allouant ce codage au composant **octets**.

NOTE 1 – Ce format de codage convient très bien pour les chaînes de caractères dans lesquelles une large gamme de caractères ISO/CEI 10646 est présente, et aucun des autres formats de codage n'est applicable ou plus utile.

NOTE 2 – L'ordre des octets du codage UTF-16BE spécifié dans Unicode, 2.6 est avec l'octet de plus fort poids en premier (le premier octet dans la chaîne des octets).

7.17.6 Le format de codage **`restricted-alphabet`** se fonde sur l'utilisation d'un alphabet restreint, choisi à partir de ceux qui sont présents dans le tableau d'alphabet restreint. Le composant **`restricted-alphabet`** doit contenir l'indice de tableau de vocabulaire de l'alphabet restreint. Ce format de codage ne peut être utilisé que pour une chaîne de caractères qui ne comporte que les caractères de l'alphabet restreint dans l'entrée de tableau d'alphabet restreint indexée par le composant **`restricted-alphabet`**. Un format de codage **`restricted-alphabet`** doit s'appliquer comme spécifié aux § 7.17.6.1 à 7.17.6.6.

7.17.6.1 Une valeur entière (commençant à zéro) doit être allouée à chaque caractère de l'alphabet restreint, dans l'ordre.

7.17.6.2 Chaque caractère de la chaîne de caractères doit être converti en un entier, qui est l'entier alloué au caractère dans l'alphabet restreint.

7.17.6.3 Chaque entier doit être représenté comme un nombre entier binaire arithmétique dans un champ binaire. La taille du champ binaire doit être déterminée par la valeur d'entier allouée au dernier caractère de l'alphabet restreint. Cette valeur d'entier doit être incrémentée de 1 pour produire une valeur entière (disons N). La taille du champ binaire doit être le nombre de bits minimal pour coder N comme un codage d'entier arithmétique.

NOTE 1 – L'incrément est nécessaire parce qu'une valeur tout à un dans le champ binaire est interprétée comme la fin de la chaîne de caractères, et ne peut pas être utilisée pour représenter un caractère. Cela signifie que si l'alphabet restreint contient un nombre de caractères qui est une exacte puissance de deux, le champ binaire aura un bit de plus qu'il ne serait autrement attendu.

NOTE 2 – Par exemple, s'il y a 24 caractères dans l'alphabet restreint, chaque caractère sera alors codé sur cinq bits, mais s'il y a 32 caractères dans l'alphabet restreint, chaque caractère sera alors codé sur six bits.

7.17.6.4 Tous ces champs binaires doivent être enchaînés (dans l'ordre) dans une chaîne binaire.

7.17.6.5 Si la longueur de la chaîne binaire résultante n'est pas un multiple entier de 8 bits, des bits '1' doivent alors être ajoutés pour faire que la longueur de la chaîne binaire soit un multiple entier de 8 bits.

7.17.6.6 La chaîne binaire résultante (qui est maintenant un multiple entier de huit bits) réinterprétée comme chaîne d'octets, doit être allouée au composant **`octets`**.

7.17.7 Le format de codage **`encoding-algorithm`** est spécifié par l'algorithme de codage (voir le § 8.3) qui est l'entrée de tableau dans le tableau d'algorithme de codage dont l'indice de tableau de vocabulaire est la valeur du composant **`encoding-algorithm`**. Le tableau d'algorithme de codage d'indice de vocabulaire doit être alloué au composant **`encoding-algorithm`**, et la chaîne d'octets résultant du codage doit être placée dans le composant **`octets`**.

8 Construction et traitement d'un document Fast Infoset

Un document Fast Infoset fait une grosse utilisation des indices de tableau de vocabulaire dans différents tableaux qui sont bâtis à différents stades de la construction et du traitement de ce document. Le paragraphe 8.1 spécifie un rangement conceptuel des composants d'une valeur abstraite du type de **`Document`** pour garantir que le créateur et les traitements des documents Fast Infoset créent des tableaux de vocabulaire identiques. Les paragraphes qui suivent spécifient les tableaux de vocabulaire qui sont bâtis et utilisés dans la création et le traitement d'un document Fast Infoset. La représentation de ces tableaux dans un système d'ordinateur est une question d'implémentation, et n'est pas normalisée. Un tableau de vocabulaire donne une transposition entre un indice de tableau de vocabulaire et des informations dans un infoset XML (qui peut être indirecte).

NOTE – Les tableaux de vocabulaire pour un document Fast Infoset sont bâtis de façon dynamique durant la construction du document Fast Infoset. Ils sont bâtis de façon dynamique à partir du contenu du document Fast Infoset durant le traitement de ce document Fast Infoset. Ils ne sont jamais échangés sous une autre forme.

8.1 Rangement conceptuel des composants d'une valeur abstraite du type `Document`

8.1.1 Afin d'assurer que des implémentations différentes allouent de la même façon les indices de tableau de vocabulaire lors de la construction et du traitement d'un document Fast Infoset, un ordre conceptuel est spécifié pour les composants d'une valeur abstraite du type **`Document`**. La construction et le traitement de telles valeurs abstraites doivent utiliser cet ordre conceptuel lors de l'ajout de chaînes (voir les § 7.13.7 et 7.14.6) et de substituts de nom (voir le § 7.16.7) aux tableaux de vocabulaire.

NOTE – Cet ordre est le même que l'ordre des codages des composants dans un document Fast Infoset. Cela n'implique pas nécessairement que la sémantique portée par le document soit traitée dans cet ordre. L'ordre n'est défini que pour garantir que le même indice de tableau de vocabulaire est alloué à toute entrée de tableau de vocabulaire donné à la fois par le créateur et le traitement d'un document Fast Infoset.

8.1.2 L'ordre conceptuel pour la construction et le traitement d'un document Fast Infoset est spécifié comme suit: les composants d'une valeur abstraite du type **`Document`** doivent être examinés conformément à l'algorithme spécifié aux § 8.1.2.1 à 8.1.2.5. L'ordre dans lequel les composants sont examinés définit l'ordre conceptuel.

8.1.2.1 Le composant de niveau supérieur de la valeur abstraite (correspondant au type **`Document`**) doit être examiné en premier.

8.1.2.2 Si le type du composant examiné est du type séquence, les composants du type séquence qui sont présents dans la valeur abstraite doivent alors être examinés dans l'ordre de leur définition textuelle, depuis le premier composant qui est présent jusqu'au dernier composant qui est présent, en appliquant les § 8.1.2.1 à 8.1.2.5 de façon récursive à chaque composant examiné.

8.1.2.3 Si le type du composant examiné est du type sequence-of, les occurrences du composant de la sequence-of doivent être examinées dans l'ordre sequence-of, depuis la première occurrence jusqu'à la dernière occurrence du composant de la sequence-of, en appliquant les § 8.1.2.1 à 8.1.2.5 de façon récursive à chaque composant examiné.

8.1.2.4 Si le type du composant examiné est du type choix, le remplaçant qui est présent dans la valeur abstraite doit être examiné et les § 8.1.2.1 à 8.1.2.5 doivent être appliqués de façon récursive à ce remplaçant.

8.1.2.5 Si le type du composant examiné est de tout autre type ASN.1, aucune autre action n'est requise pour ce composant.

8.2 Tableau d'alphabet restreint

8.2.1 Chaque document Fast Infoset a un tableau d'alphabet restreint associé. Le tableau d'alphabet restreint contient des alphabets restreints qui peuvent être référencés par un indice de tableau de vocabulaire.

8.2.2 Chaque entrée dans le tableau d'alphabet restreint doit être un ensemble ordonné de caractères ISO/CEI 10646 distincts de toute taille entre 2 et 2^{20} caractères.

NOTE – Un alphabet restreint permet un codage compact de toute chaîne de caractères consistant entièrement en caractères provenant de cet ensemble, par l'allocation d'entiers progressifs aux caractères de l'ensemble et l'utilisation de ces entiers pour coder les caractères de la chaîne (voir le § 7.17.6).

8.3 Tableau d'algorithme de codage

8.3.1 Chaque document Fast Infoset a un tableau d'algorithme de codage associé. Le tableau d'algorithme de codage contient des définitions d'algorithme de codage qui peuvent être référencées à travers un indice de tableau de vocabulaire.

8.3.2 Chaque entrée dans ce tableau spécifie le codage d'une chaîne de caractères avec des caractéristiques définies dans une chaîne d'octets (voir le § 7.17.7).

NOTE – Les caractéristiques définies peuvent se référer à la longueur de la chaîne, aux caractères qui y apparaissent, ou à un schéma de caractères arbitrairement complexe. En général, un algorithme de codage donné ne s'applique qu'à un sous-ensemble spécial et défini des chaînes de caractères de l'ISO/CEI 10646.

8.3.3 Les algorithmes de codage sont soumis aux contraintes suivantes:

a) l'algorithme de codage doit avoir un URI associé, sauf si c'est un algorithme de codage prédéfini, de sorte qu'il puisse être référencé pour ajout au tableau;

b) l'algorithme de codage doit spécifier précisément à quelles sortes de chaînes de caractères il peut s'appliquer; cette spécification doit comporter éventuellement un alphabet restreint, éventuellement une gamme de longueurs, et toutes contraintes supplémentaires sur la longueur et le contenu des chaînes de caractères (comme un canevas);

c) pour toute chaîne de caractères à laquelle il peut s'appliquer, l'algorithme de codage doit fournir une transposition réversible entre cette chaîne de caractères et une chaîne d'octets.

NOTE 1 – Les dispositions ci-dessus impliquent qu'il ne peut y avoir de chaîne de caractères S pour laquelle une étape de codage de S à E, suivie par une étape de décodage de E à S', résulte en $S' \neq S$, même si la différence entre S et S' est petite (par exemple, un ESPACE supplémentaire). D'un autre côté, il n'est pas exigé qu'une chaîne de caractères S soit codable, non plus qu'il n'est exigé que les codages soient canoniques.

NOTE 2 – Une application qui crée un document Fast Infoset à partir de données en mémoire, telles que des nombres à virgule flottante, n'est pas obligée de produire une représentation lexicale de ces données puis d'appliquer un algorithme de codage à cette représentation. L'application peut au lieu de cela créer une chaîne d'octets directement à partir de ces données, pourvu que la chaîne d'octets puisse être produite en appliquant cet algorithme de codage à une chaîne de caractères qui représente les données en mémoire et soit une chaîne de caractères à laquelle cet algorithme de codage peut s'appliquer.

NOTE 3 – Les algorithmes de codage (autres que prédéfinis) peuvent être spécifiés dans d'autres normes ou faire l'objet d'accords entre le créateur et les traitements d'un document Fast Infoset.

8.4 Tableaux de chaîne dynamiques

8.4.1 Chaque document Fast Infoset a huit tableaux de chaîne dynamiques associés. Chaque tableau de chaîne dynamique contient des chaînes de caractères qui peuvent être référencées à travers un indice de tableau de vocabulaire.

8.4.2 La présente Recommandation | Norme internationale classe toutes les chaînes de caractères qui surviennent dans un document Fast Infoset selon les huit catégories suivantes, chacune d'elles ayant un tableau de chaîne dynamique:

a) PREFIX: cette catégorie comprend des chaînes de caractères qui sont la propriété **[prefix]** d'un élément d'information **element**, **attribute** ou **namespace**.

b) NAMESPACE NAME: cette catégorie comprend des chaînes de caractères qui sont la propriété **[nom d'espace de nom]** d'un élément d'information **element, attribute** ou **namespace.**

c) LOCAL NAME: cette catégorie comprend des chaînes de caractères qui sont la propriété **[local name]** d'un élément d'information **element** ou **attribute**.

d) OTHER NCNAME: cette catégorie comprend des chaînes de caractères qui sont la propriété **[target]** d'un élément d'information **processing instruction**; ou la propriété **[name]** d'un élément d'information **unexpanded entity reference**, **unparsed entity** ou **notation**; ou la propriété **[notation name]** d'un élément d'information **unparsed entity**.

e) OTHER URI: cette catégorie comprend des chaînes de caractères qui sont la propriété **[system identifier]** ou la propriété **[public identifier]** d'un élément d'information **unexpanded entity reference**, **document type declaration, unparsed entity** ou **notation.**

f) ATTRIBUTE VALUE: cette catégorie comprend des chaînes de caractères qui sont la propriété **[normalized value]** d'un élément d'information **attribute**.

g) CONTENT CHARACTER CHUNK: cette catégorie comprend des chaînes de caractères qui sont la propriété **[character code]** d'un tronçon d'éléments d'information **character** qui sont les fils consécutifs d'un élément d'information **element** donné.

h) OTHER STRING: cette catégorie comprend des chaînes de caractères qui sont la propriété **[version]** d'un élément d'information **document**; ou la propriété **[content]** d'un élément d'information **processing instruction** ou **comment**.

8.5 Les tableaux de nom dynamiques et les substituts de nom

8.5.1 Chaque document Fast Infoset a deux tableaux de nom dynamiques associés. Chaque tableau de nom dynamique contient des substituts de nom qui peuvent être référencés à travers un indice de tableau de vocabulaire et servent à identifier un nom qualifié, qui peut être préfixé ou non préfixé (et peut avoir ou pas un nom d'espace de nom).

8.5.2 Un substitut de nom est un ensemble allant jusqu'à trois indices de tableau de vocabulaire ordonnés:

a) (facultativement) l'indice d'une chaîne dans un tableau PREFIX;

b) (facultativement) l'indice d'une chaîne dans le tableau NAMESPACE NAME;

c) l'indice d'une chaîne dans le tableau LOCAL NAME.

Le premier indice de tableau de vocabulaire ne doit pas être présent à moins que second ne soit présent.

8.5.3 Trois cas peuvent se présenter:

a) les trois indices sont présents, auquel cas le substitut de nom représente un nom qualifié préfixé;

b) seuls le second et le troisième indices sont présents, auquel cas le substitut de nom représente un nom qualifié non préfixé qui a un nom d'espace de nom;

c) seul le troisième indice est présent, auquel cas le substitut de nom représente un nom qualifié non préfixé qui n'a pas de nom d'espace de nom.

8.5.4 La présente Recommandation | Norme internationale classe tous les noms qualifiés qui peuvent survenir dans un document Fast Infoset (et donc aussi les substituts de nom qui les représentent) selon les deux catégories suivantes, chacune d'elles ayant un tableau de nom dynamique:

a) ELEMENT NAME: cette catégorie comprend les substituts de nom représentant le nom qualifié d'un élément d'information **element**.

b) ATTRIBUTE NAME: cette catégorie comprend les substituts de nom représentant le nom qualifié d'un élément d'information **attribute**.

8.5.5 Le nom qualifié qui est représenté par un substitut de nom donné doit être déterminé comme suit, selon un tableau de chaîne dynamique:

a) le premier indice de tableau de vocabulaire (s'il est présent) doit être interprété comme l'indice de tableau de vocabulaire d'une chaîne de caractères dans le tableau PREFIX;

b) le second indice de tableau de vocabulaire du substitut de nom (s'il est présent) doit être interprété comme l'indice de tableau de vocabulaire d'une chaîne de caractères dans le tableau NAMESPACE NAME;

c) le troisième entier doit être interprété comme l'indice de tableau de vocabulaire d'une chaîne de caractères dans le tableau LOCAL NAME.

9 Alphabets restreints prédéfinis

9.1 L'alphabet restreint "numérique"

9.1.1 Cet alphabet restreint a un indice de tableau de vocabulaire de 1, et comporte les quinze caractères ISO/CEI 10646 suivants (dans cet ordre):

CHIFFRE ZERO à CHIFFRE NEUF

TIRET SIGNE MOINS

SIGNE PLUS

POINT

LETTRE LATINE MINUSCULE E

ESPACE

9.1.2 Cet alphabet restreint convient pour les codages de chaînes de caractères représentant plusieurs types de nombres, y compris les nombres à virgule flottante en notation scientifique. Une seule chaîne de caractères peut contenir plusieurs nombres séparés par des espaces.

9.2 L'alphabet restreint "date et heure"

9.2.1 Cet alphabet restreint a un indice de tableau de vocabulaire de 2 et comporte les quinze caractères ISO/CEI 10646 suivants (dans cet ordre):

CHIFFRE ZERO à CHIFFRE NEUF

TIRET SIGNE MOINS

DEUX POINTS

LETTRE LATINE MAJUSCULE T

LETTRE LATINE MAJUSCULE Z

ESPACE

9.2.2 Cet alphabet restreint convient pour les codages de chaînes de caractères représentant les expressions de date et d'heure les plus communes fondées sur l'ISO 8601. Une seule chaîne de caractères peut contenir plusieurs expressions de ce type séparées par des espaces.

10 Algorithmes de codage prédéfinis

10.1 Généralités

10.1.1 Le présent paragraphe spécifie les algorithmes de codage prédéfinis. Les algorithmes de codage prédéfinis n'ont pas d'URI associé.

NOTE – Les URI sont nécessaires pour les algorithmes de codage qui doivent être identifiés explicitement dans un vocabulaire initial, mais les algorithmes de codage prédéfinis sont toujours ajoutés implicitement au tableau d'algorithme de codage et n'ont donc pas besoin d'URI.

10.1.2 Dans le présent paragraphe, le terme "mot" indique tout groupe de caractères consécutifs au sein d'une chaîne de caractères donnée, qui:

a) ne contient pas d'ESPACE;

b) est au début de la chaîne de caractères ou bien est précédée d'un ESPACE;

c) est à la fin de la chaîne de caractères ou bien est suivie par un ESPACE.

NOTE – Un "mot" n'est pas restreint à des caractères alphabétiques.

10.2 L'algorithme de codage "hexadécimal"

10.2.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 1, et ne peut s'appliquer qu'à une chaîne de caractères qui se compose des seize caractères ISO/CEI 10646 suivants:

CHIFFRE ZERO à CHIFFRE NEUF

LETTRE LATINE MAJUSCULE A à LETTRE LATINE MAJUSCULE F

et contient un nombre pair de caractères (y compris zéro).

NOTE – Les espaces blancs XML ne sont pas autorisés.

10.2.2 La chaîne de caractères doit être interprétée comme le codage hexadécimal d'une chaîne d'octets, avec le premier caractère de la chaîne qui correspond au morceau de plus fort poids du premier octet, et ainsi de suite.

10.3 L'algorithme de codage "base64"

10.3.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 2, et ne peut s'appliquer qu'à une chaîne de caractères qui:

a) se compose entièrement des caractères LETTRE LATINE MAJUSCULE A à LETTRE LATINE MAJUSCULE Z, LETTRE LATINE MINUSCULE A à LETTRE LATINE MINUSCULE Z, CHIFFRE ZERO à CHIFFRE NEUF, SIGNE PLUS, BARRE OBLIQUE, et SIGNE EGAL; et

NOTE – Ceci ne permet pas la présence d'espaces blancs XML dans la chaîne de caractères.

b) est une instance valide du codage de transfert de contenu spécifié dans le RFC 2045, 6.8 de l'IETF, ou pourrait devenir une instance valide de ce codage par l'insertion d'espaces blancs XML lorsqu'elle est demandée par le document RFC 2045 de l'IETF.

10.3.2 La chaîne de caractères doit être interprétée comme le codage Base64 (voir le document RFC 2045 de l'IETF) d'une chaîne d'octets (en supposant que cet espace blanc XML supplémentaire est présent là où il est exigé par le document RFC 2045 de l'IETF). La chaîne d'octets résultante est la chaîne d'octets spécifiée par ce codage Base64.

10.3.3 Cet algorithme de codage convient pour les chaînes de caractères de codage qui ne contiennent pas d'espace blanc XML, et sont des chaînes Base64 (de toute longueur) ou deviendraient des chaînes Base64 avec l'ajout d'espaces blancs XML.

10.4 L'algorithme de codage "court"

10.4.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 3, et ne peut s'appliquer qu'à une chaîne de caractères qui satisfait toutes les conditions suivantes:

a) la chaîne de caractères se compose entièrement des caractères CHIFFRE ZERO à CHIFFRE NEUF, TIRET SIGNE MOINS, et ESPACE;

b) ni le premier ni le dernier caractère de la chaîne de caractères n'est un ESPACE, et il n'y a aucune paire d'ESPACE adjacents;

c) la chaîne de caractères contient au moins un mot (voir le § 10.1.2);

d) chaque TIRET SIGNE MOINS qui est présent est le premier caractère d'un mot;

e) chaque TIRET SIGNE MOINS est suivi par au moins un caractère dans la gamme CHIFFRE UN à CHIFFRE NEUF;

f) chaque CHIFFRE ZERO est le seul caractère d'un mot ou bien est précédé par un caractère dans la gamme CHIFFRE UN à CHIFFRE NEUF;

g) chaque mot dans la chaîne de caractères, s'il est interprété comme une chaîne de caractères numériques d'entiers algébriques de base 10, donne une valeur dans la gamme –32768 à 32767.

10.4.2 Chaque mot (voir le § 10.1.2) dans la chaîne de caractères doit être interprété comme une chaîne de caractères numériques d'entiers algébriques de base 10 et doit être représenté comme un entier complément à deux de 16 bits.

10.4.3 Chaque groupe de huit bits dans l'entier complément à deux de 16 bits pour un mot doit produire un octet de la chaîne d'octets, commençant par le groupe de huit bits d'ordre le plus élevé. Le bit d'ordre le plus élevé de chaque groupe de huit bits doit devenir le bit de plus fort poids de l'octet correspondant. S'il y a plusieurs mots dans la chaîne de caractères, ils doivent être codés dans l'ordre, et les octets produits par les divers entiers compléments à deux de 16 bits doivent être enchaînés dans cet ordre.

10.4.4 Cet algorithme de codage convient pour le codage de chaînes de caractères représentant un seul entier dans la gamme –32768 à 32767 (représentables comme entiers compléments à deux de 16 bits) ou une liste de tels entiers.

10.5 L'algorithme de codage "int"

10.5.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 4, et ne peut s'appliquer qu'à une chaîne de caractères qui satisfait à toutes les conditions suivantes:

a) la chaîne de caractères satisfait aux conditions spécifiées au § 10.4.1 a à f; et

b) chaque mot (voir le § 10.1.2) dans la chaîne de caractères, s'il est interprété comme une chaîne de caractères numériques d'entiers algébriques de base 10, donne une valeur dans la gamme –2147483648 à 2147483647.

10.5.2 Chaque mot (voir le § 10.1.2) dans la chaîne de caractères doit être interprété comme une chaîne de caractères numériques d'entiers algébriques de base 10 et doit être représenté comme un entier complément à deux de 32 bits.

10.5.3 Chaque groupe de huit bits dans l'entier complément à deux de 32 bits pour un mot doit produire un octet de la chaîne d'octets, commençant par le groupe de huit bits d'ordre le plus élevé. Le bit d'ordre le plus élevé dans chaque groupe de huit bits doit devenir le bit de plus fort poids de l'octet correspondant. S'il y a plusieurs mots dans la chaîne de caractères, ils doivent être codés dans l'ordre, et les octets produits par les divers entiers compléments à deux de 32 bits doivent être concaténés dans cet ordre.

10.5.4 Cet algorithme de codage convient pour le codage de chaînes de caractères représentant un seul entier dans la gamme –2147483648 à 2147483647 (représentable comme un entier complément à deux de 32 bits) ou une liste de tels entiers.

10.6 L'algorithme de codage "long"

10.6.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 5, et ne peut s'appliquer qu'à une chaîne de caractères qui satisfait à toutes les conditions suivantes:

a) la chaîne de caractères satisfait les conditions spécifiées au § 10.4.1 a à f;

b) chaque mot (voir le § 10.1.2) de la chaîne de caractères, s'il est interprété comme une chaîne de caractères numériques d'entiers algébriques de base 10, donne une valeur dans la gamme –9223372036854775808 à 9223372036854775807.

10.6.2 Chaque mot (voir le § 10.1.2) de la chaîne de caractères doit être interprété comme une chaîne de caractères numériques d'entiers algébriques de base 10 et doit être représenté comme un entier complément à deux de 64 bits.

10.6.3 Chaque groupe de 8 bits dans l'entier complément à deux de 64 bits pour un mot doit produire un octet de la chaîne d'octets, commençant par le groupe de huit bits d'ordre le plus élevé. Le bit d'ordre le plus élevé dans chaque groupe de 8 bits doit devenir le bit de plus fort poids de l'octet correspondant. S'il y a plusieurs mots dans la chaîne de caractères, ils doivent être codés dans l'ordre, et les octets produits par les divers entiers compléments à deux de 64 bits doivent être concaténés dans cet ordre.

10.6.4 Cet algorithme de codage convient pour le codage de chaînes de caractères représentant un seul entier dans la gamme –9223372036854775808 à 9223372036854775807 (représentable comme entier complément à deux de 64 bits) ou une liste de tels entiers.

10.7 L'algorithme de codage "booléen"

10.7.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 6, et ne peut s'appliquer qu'à une chaîne de caractères qui satisfait à toutes les conditions suivantes:

a) la chaîne de caractères se compose entièrement d'un ou plusieurs des mots "Faux" ou "Vrai", et du caractère ESPACE;

b) ni le premier ni le dernier caractère de la chaîne de caractères n'est un ESPACE, et il n'y a pas de paire de caractères ESPACE adjacents;

c) la chaîne de caractères contient au moins un mot (voir le § 10.1.2).

10.7.2 Chaque mot "Faux" ou "Vrai" dans la chaîne de caractères doit être codé comme un seul bit (mis respectivement à zéro ou un) de la chaîne d'octet produite, commençant au cinquième bit du premier octet et continuant jusqu'au huitième bit du premier octet. Les bits suivants sont placés dans les octets suivants en procédant du premier bit de chaque octet jusqu'au huitième bit de cet octet, en n'utilisant que le nombre d'octets requis. Tout bit inutilisé dans le dernier octet doit être mis à zéro.

10.7.3 Les quatre premiers bits du premier octet doivent contenir le nombre de bits inutilisés du dernier octet, codés comme entier arithmétique de 4 bits.

NOTE – Le premier octet peut aussi être le dernier octet et contenir jusqu'à trois bits inutilisés. S'il y a plus d'un octet, le dernier octet peut contenir jusqu'à sept bits inutilisés.

10.8 L'algorithme de codage "float"

10.8.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 7, et ne peut s'appliquer qu'à une chaîne de caractères qui satisfait à toutes les conditions suivantes:

a) la chaîne de caractères se compose entièrement des caractères CHIFFRE ZERO à CHIFFRE NEUF, TIRET SIGNE MOINS, POINT, LETTTRE LATINE MAJUSCULE E, et ESPACE;

NOTE – La LETTRE LATINE MINUSCULE E n'est pas admise, car le codage ne serait alors pas réversible.

b) ni le premier ni le dernier caractère de la chaîne de caractères n'est un ESPACE, et il n'y a pas de paire de caractères ESPACE adjacents;

c) la chaîne de caractères contient au moins un mot (voir le § 10.1.2);

d) chaque mot de la chaîne de caractères correspond à la représentation lexicale canonique d'un nombre à virgule flottante en précision simple, comme spécifié dans le schéma XML du W3C, Partie 2, 3.2.4;

e) chaque mot de la chaîne de caractères, s'il est interprété comme une chaîne de caractères numérique de base 10 à virgule flottante, donne une valeur qui peut être représentée comme un nombre à virgule flottante en précision simple de 32 bits de l'IEEE 754.

10.8.2 Chaque mot (voir le § 10.1.2) de la chaîne de caractères doit être interprété comme une chaîne de caractères numérique de base 10 à virgule flottante et doit être représentée comme un nombre à virgule flottante en précision simple de 32 bits de l'IEEE 754.

10.8.3 Chaque groupe de 8 bits dans le nombre à virgule flottante en précision simple de 32 bits de l'IEEE 754 pour un mot doit produire un octet de la chaîne d'octets, commençant par le groupe de 8 bits de tête. Le bit de tête de chaque groupe de 8 bits doit devenir le bit de plus fort poids de l'octet correspondant. S'il y a plusieurs mots dans la chaîne de caractères, ils doivent être codés dans l'ordre, et les octets produits par les différents nombres à virgule flottante en précision simple de 32 bits de l'IEEE 754 doivent être concaténés dans cet ordre.

10.8.4 Cet algorithme de codage convient pour le codage de chaînes de caractères représentant un seul nombre à virgule flottante représentable comme nombre à virgule flottante en précision simple de 32 bits de l'IEEE 754 ou une liste de tels nombres à virgule flottante.

10.9 L'algorithme de codage "double"

10.9.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 8, et ne peut s'appliquer qu'à une chaîne de caractères qui satisfait à toutes les conditions suivantes:

a) la chaîne de caractères satisfait aux conditions spécifiées au § 10.8.1 a à c;

b) chaque mot de la chaîne de caractères correspond à la représentation lexicale canonique d'un nombre à virgule flottante avec double extension comme spécifié dans le schéma XML du W3C, Partie 2, 3.2.5;

c) chaque mot (voir le § 10.1.2) de la chaîne de caractères, s'il est interprété comme une chaîne de caractères numérique de base 10 à virgule flottante, donne une valeur qui peut être représentée comme un nombre à virgule flottante avec double extension de 64 bits de l'IEEE 754.

10.9.2 Chaque mot (voir le § 10.1.2) de la chaîne de caractères doit être interprété comme une chaîne de caractères numérique de base 10 à virgule flottante et doit être représenté comme un nombre à virgule flottante avec double extension de 64 bits de l'IEEE 754.

10.9.3 Chaque groupe de 8 bits dans le nombre à virgule flottante de 64 bits de l'IEEE 754 pour un mot doit produire un octet de la chaîne d'octets, commençant par le groupe de 8 bits de tête. Le bit de tête de chaque groupe de 8 bits doit devenir le bit de plus fort poids de l'octet correspondant. S'il y a plusieurs mots dans la chaîne de caractères, ils doivent être codés dans l'ordre, et les octets produits par les divers nombres à virgule flottante de 64 bits de l'IEEE 754 doivent être concaténés.

10.9.4 Cet algorithme de codage convient pour le codage de chaînes de caractères représentant un seul nombre à virgule flottante pouvant être représenté par un nombre à virgule flottante avec double extension de 64 bits de l'IEEE 754 ou une liste de tels nombres à virgule flottante.

10.10 L'algorithme de codage "uuid"

10.10.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 9, et ne peut s'appliquer qu'à une chaîne de caractères qui satisfait à toutes les conditions suivantes:

a) la chaîne de caractères se compose entièrement des caractères CHIFFRE ZERO à CHIFFRE NEUF, LETTRE LATINE MINUSCULE A à LETTRE LATINE MINUSCULE F, TIRET SIGNE MOINS, et ESPACE;

b) ni le premier ni le dernier caractère de la chaîne de caractères n'est un ESPACE, et il n'y a pas de paire de caractères ESPACE adjacents;

c) la chaîne de caractères contient au moins un mot (voir le § 10.1.2);

d) chaque mot contient exactement 36 caractères;

e) dans chaque mot, il y a exactement quatre caractères TIRET SIGNE MOINS, occupant les positions 9, 14, 19 et 24 (en comptant à partir de un).

10.10.2 Chaque mot (voir le § 10.1.2) de la chaîne de caractères doit être interprété comme la représentation hexadécimale d'une UUID (voir la Rec. UIT-T X.667 | ISO/CEI 9834-8, § 6.4), et doit être représenté comme un entier arithmétique de 16 octets comme spécifié dans la Rec. UIT-T X.667 | ISO/CEI 9834-8, au § 6.3. S'il y a plusieurs mots, les différents entiers arithmétiques de 16 octets doivent être concaténés.

10.10.3 Cet algorithme de codage convient pour le codage de chaînes de caractères représentant un seul UUID ou une liste d'UUID.

10.11 L'algorithme de codage "cdata"

10.11.1 Cet algorithme de codage a un indice de tableau de vocabulaire de 10, et peut s'appliquer à toute chaîne de caractères.

10.11.2 La chaîne d'octets produite doit être le codage UTF-8 (voir l'ISO/CEI 10646) de la chaîne de caractères.

10.11.3 Cet algorithme ne doit être utilisé qu'avec les infosets XML créés par l'analyse grammaticale d'un document XML et où des informations supplémentaires identifient que la chaîne de caractères correspond à une section CDATA complète (voir W3C XML 1.0 et W3C XML 1.1). Si cet algorithme de codage est utilisé au sein d'un document Fast Infoset, toutes les chaînes de caractères qui correspondent aux sections CDATA doivent alors se voir appliquer cet algorithme de codage.

11 Restrictions sur les infosets XML pris en charge et autres simplifications

11.1 La présente Recommandation | Norme internationale prend en charge la plupart des infosets XML dont la probabilité d'occurrence pratique est non négligeable, mais ne prend pas en charge certains infosets XML théoriquement possibles qui ne surviennent normalement pas.

11.2 Le terme "cohérent avec XML" est utilisé dans le présent paragraphe avec la signification suivante: un ensemble de propriétés d'un ou plusieurs éléments d'information est "cohérent avec XML" si cet ensemble de propriétés peut avoir été obtenu par l'analyse grammaticale d'un document XML à espace de nom bien conformé convenable.

11.3 Les infosets XML qui sont pris en charge doivent satisfaire à toutes les conditions suivantes:

a) la propriété **[all declarations processed]** de l'élément d'information **document** a la valeur **vraie**;

b) la propriété **[in-scope namespaces]** de chaque élément d'information **element** forme conjointement avec la propriété **[namespace attributes]** de tout élément d'information **element** un ensemble cohérent avec XML;

c) la propriété **[namespace name]** de chaque élément d'information **element** forme conjointement avec la propriété **[namespace attributes]** de tout élément d'information **element** et la propriété **[prefix]** de cet élément d'information **element** un ensemble cohérent avec XML;

d) la propriété **[namespace name]** de chaque élément d'information **attribute** forme conjointement avec la propriété **[namespace attributes]** de tout élément d'information **element** et la propriété **[prefix]** de cet élément d'information **attribute** un ensemble cohérent avec XML;

e) la propriété **[references]** de chaque élément d'information **attribute** forme conjointement avec la propriété **[normalized value]** de l'élément d'information **attribute** un ensemble cohérent avec XML;

f) la propriété **[notation]** de chaque élément d'information **processing instruction** forme conjointement avec la propriété **[target]** de l'élément d'information **processing instruction** et la propriété **[notations]** de l'élément d'information **document** un ensemble cohérent avec XML;

g) la propriété **[notation]** de chaque élément d'information **unparsed entity** forme conjointement avec la propriété **[notation name]** de l'élément d'information **unparsed entity** et la propriété **[notations]** de l'élément d'information **document** un ensemble cohérent avec XML;

h) la propriété **[element content whitespace]** de tout élément d'information **character** qui ne représente pas un espace blanc a la valeur **faux**;

i) la propriété **[element content whitespace]** de chaque élément d'information **character** conjointement avec la propriété **[character code]** de l'élément d'information **character** forme un ensemble cohérent avec XML;

j) la propriété **[normalized value]** de tout élément d'information **attribute** et la propriété **[content]** de tout élément d'information **comment** et **processing** instruction contient au plus 2^{32} caractères.

11.4 Les propriétés suivantes des éléments d'information de l'Ensemble d'informations XML ne sont pas incluses dans les types ASN.1 représentant ces éléments d'information:

a) les propriétés **[document element]**, **[base URI]** et **[all declarations processed]** de l'élément d'information **document** (voir les § 7.2.30, 7.2.31 et 7.2.32);

b) les propriétés **[in-scope namespaces]**, **[base URI]** et **[parent]** de l'élément d'information **element** (voir les § 7.3.8, 7.3.9 et 7.3.10);

c) les propriétés **[specified]**, **[attribute type]**, **[references]** et **[owner element]** de l'élément d'information **attribute** (voir les § 7.4.7, 7.4.8, 7.4.9 et 7.4.10);

d) les propriétés **[notation]** et **[parent]** de l'élément d'information **processing instruction** (voir les § 7.5.7 et 7.5.8);

e) les propriétés **[declaration base URI]** et **[parent]** de l'élément d'information **unexpanded entity reference** (voir les § 7.6.7 et 7.6.8);

f) la propriété **[element content whitespace]** de l'élément d'information **character** (voir le § 7.7.7);

g) la propriété **[parent]** de l'élément d'information **character** (voir le § 7.7.8);

h) la propriété **[parent]** de l'élément d'information **comment** (voir le § 7.8.6);

i) la propriété **[parent]** de l'élément d'information **document type declaration** (voir le § 7.9.7);

j) les propriétés **[declaration base URI]** et **[notation]** de l'élément d'information **unparsed entity** (voir les § 7.10.8 et 7.10.9);

k) la propriété **[declaration base URI]** de l'élément d'information **notation** (voir le § 7.11.7).

12 Codage de niveau binaire du type `Document`

12.1 Le présent paragraphe spécifie les codages spéciaux du type `Document` pour former un document Fast Infoset.

NOTE – Ces codages spéciaux sont conçus pour optimiser la vitesse de traitement et la densité, qui sont considérées comme critiques dans de nombreuses utilisations attendues de la présente Recommandation | Norme internationale.

12.2 Les codages sont spécifiés en termes d'actions que doit accomplir le codeur, qui ont pour résultat d'ajouter des bits à un flux binaire. Le flux binaire initial est vide ou comporte une déclaration XML (voir le § 12.3).

12.3 Une déclaration XML (voir W3C XML 1.1, 2.8) peut (comme option du créateur) être incluse au début du flux binaire. La déclaration XML (si présente) doit être une des chaînes de caractères suivantes, codée en UTF-8:

1) **<?xml encoding='finf'?>**
2) **<?xml encoding='finf' standalone='yes'?>**
3) **<?xml encoding='finf' standalone='no'?>**
4) **<?xml version='1.0' encoding='finf'?>**
5) **<?xml version='1.0' encoding='finf' standalone='yes'?>**
6) **<?xml version='1.0' encoding='finf' standalone='no'?>**
7) **<?xml version='1.1' encoding='finf'?>**
8) **<?xml version='1.1' encoding='finf' standalone='yes'?>**
9) **<?xml version='1.1' encoding='finf' standalone='no'?>**

12.4 Le numéro de version (s'il est présent) dans la déclaration XML doit être réglé à la propriété **[version]** correspondante de l'élément d'information **document**. La déclaration XML ne doit pas inclure de numéro de version si la propriété **[version]** n'a pas de valeur.

12.5 La déclaration autonome (si elle est présente) dans la déclaration XML doit être réglée à la propriété **[standalone]** correspondante de l'élément d'information **document**. La déclaration XML ne doit pas inclure de déclaration autonome si la propriété **[standalone]** n'a pas de valeur.

12.6 Les seize bits '1110000000000000' doivent alors être ajoutés au flux binaire.

NOTE – Ces bits vont survenir au début du document Fast Infoset ou vont suivre la déclaration XML. En l'absence de déclaration XML, un analyseur peut distinguer, en examinant les 16 premiers bits d'un codage, un document Fast Infoset potentiel de tout autre document XML 1.0 W3C ou XML 1.1 W3C bien conformé, parce que ces 16 bits ne peuvent jamais survenir au début d'un document XML bien conformé.

12.7 Un champ binaire de seize bits contenant le numéro de version de la présente Recommandation | Norme internationale (voir le § 12.9) codé comme un entier arithmétique de 16 bits doit alors être ajouté au flux binaire.

12.8 Le bit '0' (bourrage) doit alors être ajouté au flux binaire.

NOTE – Ceci est fait pour garantir l'alignement des octets dans les parties ultérieures du codage.

12.9 Le numéro de version de cette édition de la présente Recommandation | Norme internationale est 1.

NOTE – Des éditions ultérieures de la présente Recommandation | Norme internationale sont attendues, qui augmenteront le numéro de version car des problèmes d'interfonctionnement entre la nouvelle édition et les éditions antérieures surviendront vraisemblablement.

12.10 Le codage ECN (voir la Rec. UIT-T X.692 | ISO/CEI 8825-3) de la valeur abstraite du type **`Document`**, spécifié par le composant Module de liaison de codage au § A.2, doit être ajouté au flux binaire.

NOTE – L'Annexe C donne une description informelle des codages spécifiés au § A.2.

12.11 Si le codage de la valeur abstraite du type **`Document`** ne se termine pas sur le dernier bit d'un octet, les quatre bits '0000' (bourrage) doivent alors être ajoutés au flux binaire, pour compléter le dernier octet.

12.12 Une fois achevées les étapes ci-dessus, le contenu du flux binaire est un document Fast Infoset.

Annexe A

Module ASN.1 et modules ECN pour documents Fast Infoset

(La présente annexe fait partie intégrante de la présente Recommandation | Norme internationale)

A.1 Définition du module ASN.1

```
FastInfoset {joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0)
modules(0) fast-infoset(0)}

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

finf-doc-opt-decl OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) asn1(1)
            generic-applications(10) fast-infoset(0) encodings(1)
            optional-xml-declaration(0)}

finf-doc-no-decl OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) asn1(1)
            generic-applications(10) fast-infoset(0) encodings(1)
            no-xml-declaration(1)}

Document ::= SEQUENCE {
      additional-data         SEQUENCE (TAILLE(1..one-meg)) DE
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } FACULTATIF,
      initial-vocabulary      SEQUENCE {
            external-vocabulary         URI FACULTATIF,
            restricted-alphabets        SEQUENCE (TAILLE(1..256)) DE
                  NonEmptyOctetString FACULTATIF,
            encoding-algorithms         SEQUENCE (TAILLE(1..256)) DE
                  NonEmptyOctetString FACULTATIF,
            prefixes                    SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            namespace-names             SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            local-names                 SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            other-ncnames               SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            other-uris                  SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            attribute-values            SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            content-character-chunks    SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            other-strings               SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            element-name-surrogates     SEQUENCE (TAILLE(1..one-meg)) DE
                  NameSurrogate FACULTATIF,
            attribute-name-surrogates   SEQUENCE (TAILLE(1..one-meg)) DE
                  NameSurrogate FACULTATIF }
                  (CONSTRAINED BY {
                        -- Si le composant initial-vocabulary est présent, au moins
                        -- un de ses composants doit être présent -- }) FACULTATIF,
      notations                     SEQUENCE (TAILLE(1..MAX)) DE
                                          Notation FACULTATIF,
      unparsed-entities             SEQUENCE (TAILLE(1..MAX)) DE
                                          UnparsedEntity FACULTATIF,
      character-encoding-scheme     NonEmptyOctetString FACULTATIF,
      standalone                    BOOLEAN FACULTATIF,
      version                       NonIdentifyingStringOrIndex FACULTATIF
                                      -- catégorie AUTRE CHAINE --,
      children                      SEQUENCE (TAILLE(0..MAX)) DE
            CHOIX {
                  element                   Element,
                  processing-instruction    ProcessingInstruction,
                  comment                   Comment,
                  document-type-declaration DocumentTypeDeclaration }}
```

```
one-meg INTEGER ::= 1048576 -- Deux à la puissance 20

four-gig INTEGER ::= 4294967296 -- Deux à la puissance 32

NonEmptyOctetString ::= OCTET STRING (TAILLE(1..four-gig))

URI ::= NonEmptyOctetString

Element ::= SEQUENCE {
      namespace-attributes    SEQUENCE (TAILLE(1..MAX)) DE
            NamespaceAttribute FACULTATIF,
      qualified-name          QualifiedNameOrIndex
                                    --  catégorie ELEMENT NAME --,
      attributes              SEQUENCE (TAILLE(1..MAX)) DE
            Attribute FACULTATIF,
      children                SEQUENCE (TAILLE(0..MAX)) DE
            CHOIX {
                  element                       Element,
                  processing-instruction        ProcessingInstruction,
                  unexpanded-entity-reference   UnexpandedEntityReference,
                  character-chunk               CharacterChunk,
                  comment                       Comment }}

Attribute ::= SEQUENCE {
      qualified-name      QualifiedNameOrIndex
                                    -- catégorie ATTRIBUTE NAME --,
      normalized-value    NonIdentifyingStringOrIndex
                                    -- catégorie ATTRIBUTE VALUE -- }

ProcessingInstruction ::= SEQUENCE {
      target       IdentifyingStringOrIndex
                        -- catégorie OTHER NCNAME --,
      content      NonIdentifyingStringOrIndex
                        -- catégorie OTHER STRING -- }

UnexpandedEntityReference ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                    -- catégorie OTHER NCNAME --,
      system-identifier     IdentifyingStringOrIndex FACULTATIF
                                    -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                    -- catégorie OTHER URI -- }

CharacterChunk ::= SEQUENCE {
      character-codes             NonIdentifyingStringOrIndex
                                          -- catégorie CONTENT CHARACTER CHUNK -- }

Comment ::= SEQUENCE {
      content     NonIdentifyingStringOrIndex -- catégorie OTHER STRING --}

DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier     IdentifyingStringOrIndex FACULTATIF
                                    -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                    -- catégorie OTHER URI --,
      children              SEQUENCE (TAILLE(0..MAX)) DE
            ProcessingInstruction }

UnparsedEntity ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                    -- catégorie OTHER NCNAME --,
      system-identifier     IdentifyingStringOrIndex
                                    -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                    -- catégorie OTHER URI --,
      notation-name         IdentifyingStringOrIndex
                                    -- catégorie OTHER NCNAME -- }

Notation ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                    -- catégorie OTHER NCNAME --,
      system-identifier     IdentifyingStringOrIndex FACULTATIF
                                    -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                    -- catégorie OTHER URI -- }
```

```
NamespaceAttribute ::= SEQUENCE {
      prefix             IdentifyingStringOrIndex FACULTATIF
                                 -- catégorie PREFIX --,
      namespace-name     IdentifyingStringOrIndex FACULTATIF
                                 -- catégorie NAMESPACE NAME -- }

IdentifyingStringOrIndex ::= CHOIX {
      literal-character-string   NonEmptyOctetString,
      string-index               ENTIER (1..one-meg) }

NonIdentifyingStringOrIndex ::= CHOIX {
      literal-character-string    SEQUENCE {
            add-to-table               BOOLEAN,
            character-string           EncodedCharacterString },
      string-index                ENTIER (0..one-meg) }

NameSurrogate ::= SEQUENCE {
      prefix-string-index           ENTIER(1..one-meg) FACULTATIF,
      namespace-name-string-index   ENTIER(1..one-meg) FACULTATIF,
      local-name-string-index       ENTIER(1..one-meg) }
      (CONTRAINT PAR   {-- prefix-string-index ne doit être présent que si
                        -- namespace-name-string-index est présent --})

QualifiedNameOrIndex ::= CHOIX {
      literal-qualified-name  SEQUENCE {
            prefix                IdentifyingStringOrIndex FACULTATIF
                                        -- catégorie PREFIX --,
            namespace-name        IdentifyingStringOrIndex FACULTATIF
                                        -- catégorie NAMESPACE NAME --,
            local-name            IdentifyingStringOrIndex
                                        -- catégorie LOCAL NAME -- },
      name-surrogate-index    ENTIER (1..one-meg) }

EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOIX {
            utf-8                  NULL,
            utf-16                 NULL,
            restricted-alphabet    ENTIER(1..256),
            encoding-algorithm     ENTIER(1..256) },
      octets                NonEmptyOctetString }

END
```

A.2 Définitions de modules ECN

```
FastInfosetEDM {joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0)
modules(0) fast-infoset-edm(1)}
      ENCODING-DEFINITIONS ::= BEGIN
      EXPORTS FastInfosetEncodingSet;
      RENAMES
              #INTEGER AS #PositiveOrNonNegativeInteger
              IN #IdentifyingStringOrIndex.string-index,
                 #NonIdentifyingStringOrIndex.string-index,
                 #QualifiedNameOrIndex.name-surrogate-index,
                 #NameSurrogate.namespace-name-string-index,
                 #NameSurrogate.prefix-string-index,
                 #NameSurrogate.local-name-string-index
              FROM FastInfoset;

      /* RENAMES automatically imports:
      #Document, #NonEmptyOctetString, #NameSurrogate, #ProcessingInstruction,
      #UnexpandedEntityReference, #Comment, #DocumentTypeDeclaration,
      #UnparsedEntity, #Notation, #Element, #Attribute, #CharacterChunk,
      #NamespaceAttribute, #IdentifyingStringOrIndex, #NonIdentifyingStringOrIndex,
      #QualifiedNameOrIndex, #EncodedCharacterString FROM FastInfoset;
      */
```

```
-- Classes de codage utiles
#PositiveOrNonNegativeInteger ::= #ENTIER

#NonEmptySequenceOfLength ::= #ENTIER(1..1048576)

#NonEmptyOctetStringLength ::= #ENTIER(1..4294967296)

#TwoAlternativeDiscriminant ::= #ENTIER(0..1)

#ThreeAlternativeDiscriminant ::= #ENTIER(0..2)

#FourAlternativeDiscriminant ::= #ENTIER(0..3)

#FiveAlternativeDiscriminant ::= #ENTIER(0..4)

-- Utilisé lors du codage de la longueur d'une SEQUENCE DE (voir le § C.21)

#NonEmptySequenceOfLengthAlternatives1 ::= #ALTERNATIVES {
      small        #ENTIER(1..128),
      large        #ENTIER(129..1048576) }
-- Utilisé lors du codage de la longueur d'une NonEmptyOctetString (voir le § C.22)

#NonEmptyOctetStringLengthAlternatives2 ::= #ALTERNATIVES {
      small        #INT(1..64),
      medium       #INT(65..320),
      large        #INT(321..4294967296) }

-- Utilisé lors du codage de la longueur d'une NonEmptyOctetString (voir le § C.23)

#NonEmptyOctetStringLengthAlternatives5 ::= #ALTERNATIVES {
      small        #INT(1..8),
      medium       #INT(9..264),
      large        #INT(265..4294967296) }

-- Utilisé lors du codage de la longueur d'une NonEmptyOctetString (voir le § C.24)

#NonEmptyOctetStringLengthAlternatives7 ::= #ALTERNATIVES {
      small        #INT(1..2),
      medium       #INT(3..258),
      large        #INT(259..4294967296) }

-- Utilisé lors du codage d'un entier positif (voir le § C.25)

#PositiveIntegerAlternatives2 ::= #ALTERNATIVES {
      small         #INT(1..64),
      medium        #INT(65..8256),
      large         #INT(8257..1048576) }

-- Utilisé lors du codage d'un entier positif (voir le § C.27)

#PositiveIntegerAlternatives3 ::= #ALTERNATIVES {
      small         #INT(1..32),
      medium        #INT(33..2080),
      medium-large  #INT(2081..526368),
      large         #INT(526369..1048576) }

-- Utilisé lors du codage d'un entier positif (voir le § C.28)

#PositiveIntegerAlternatives4 ::= #ALTERNATIVES {
      small         #INT(1..16),
      medium        #INT(17..1040),
      medium-large  #INT(1041..263184),
      large         #INT(263185..1048576) }

-- Utilisé lors du codage d'un entier non-négatif (voir le § C.26)

#NonNegativeIntegerAlternatives2 ::= #ALTERNATIVES {
      zero          #INT(0),
      small         #INT(1..64),
      medium        #INT(65..8256),
      large         #INT(8257..1048576) }

-- Utilisé pour insérer le prébourrage avant codage dans de nombreux cas

#PrecededByPrepadding{<#C>} ::= #CONCATENATION {
      prepadding      #PAD,
      original        #C }
```

```
-- Utilisé pour insérer un discriminant à deux alternatives avant un codage

#PrecededByTwoAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      discriminant    #TwoAlternativeDiscriminant,
      original        #C }

-- Utilisé pour insérer un discriminant à trois alternatives avant un codage

#PrecededByThreeAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      discriminant    #ThreeAlternativeDiscriminant,
      original        #C }

-- Utilisé pour insérer un discriminant à quatre alternatives avant un codage

#PrecededByFourAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      prepadding      #PAD,
      discriminant    #FourAlternativeDiscriminant,
      original        #C }

-- Utilisé pour insérer un discriminant à cinq alternatives avant un codage

#PrecededByFiveAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      prepadding      #PAD,
      discriminant    #FiveAlternativeDiscriminant,
      original        #C }

-- Utilisé pour insérer un champ longueur avant le codage d'une SEQUENCE OF

#PrecededByNonEmptySequenceOfLength{<#C>} ::= #CONCATENATION {
      length          #NonEmptySequenceOfLength,
      original        #C }

-- Utilisé pour insérer un champ longueur avant le codage d'une NonEmptyOctetString

#PrecededByNonEmptyOctetStringLength{<#C>} ::= #CONCATENATION {
      length          #NonEmptyOctetStringLength,
      original        #C }

-- Code un élément du composant notations du type Document
-- (voir le § C.2.6.1)

eNotationDriver1 #Notation ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNotationPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Code un élément du composant unparsed-entities du type Document
-- voir le § C.2.7.1)

eUnparsedEntityDriver1 #UnparsedEntity ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eUnparsedEntityPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Code un élément du composant namespace-attributes du type Element
-- (voir le § C.3.4.2)

eNamespaceAttributeDriver1 #NamespaceAttribute ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNamespaceAttributePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Code un élément du composant attributes du type Element
-- (voir le § C.3.6.1)

eAttributeDriver1 #Attribute ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eAttributePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Code un élément des composant attribute-values,
-- content-character-chunks, et other-strings du type Document
-- (voir le § C.2.5.4)

eEncodedCharacterStringDriver1 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eEncodedCharacterStringPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }
```

```
-- Code un élément des composants element-name-surrogates et
-- attribute-name-surrogates du type Document (voir le § C.2.5.5)

eNameSurrogateDriver1 #NameSurrogate ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNameSurrogatePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Code le composant initial-vocabulary du type Document
-- (voir le § C.2.5)

eInitialVocabularyPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eInitialVocabularyWithPrepadding1 }

-- Insère le prébourrage avant le codage d'un élément du composant
-- notations du type Document (voir le § C.2.6.1)

eNotationPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNotationWithPrepadding1 }

-- Insère le prébourrage avant le codage d'un élément du composant
-- unparsed-entities du type Document(voir le § C.2.7.1)

eUnparsedEntityPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eUnparsedEntityWithPrepadding1 }

-- Insère le prébourrage avant le codage d'un élément du composant standalone
-- du type Document (voir le § C.2.9)

eStandalonePrepaddingAdder1 #BOOL ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eStandaloneWithPrepadding1 }

-- Insère le prébourrage avant le codage d'un élément du composant
-- children du type DocumentTypeDeclaration (voir le § C.9.6)

eDocTypeDeclChildPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eDocTypeDeclChildWithPrepadding1 }

-- Insère le prébourrage avant le codage d'un élément du composant
-- namespace-attributes du type Element (voir le § C.3.4.2)

eNamespaceAttributePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNamespaceAttributeWithPrepadding1 }

-- Insère le prébourrage avant le codage d'un élément du composant
-- attributes du type Element (voir le § C.3.6.1)

eAttributePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eAttributeWithPrepadding1 }

-- Insère le prébourrage avant le codage du composant
-- literal-qualified-name du type QualifiedNameOrIndex.
-- Utilisé lorsque le codage débute sur le second bit d'un octet
-- (voir le § C.17.3)

eLiteralQualifiedNamePrepaddingAdder2 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eLiteralQualifiedNameWithPrepadding2 }

-- Insère le prébourrage avant le codage du composant
-- literal-qualified-name du type QualifiedNameOrIndex.
-- Utilisé lorsque le codage débute sur le troisième bit d'un octet
-- (voir le § C.18.3)

eLiteralQualifiedNamePrepaddingAdder3 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eLiteralQualifiedNameWithPrepadding3 }
```

```
-- Insère le prébourrage avant le codage d'un élément des composants
-- restricted-alphabets, encoding-algorithms, prefixes, namespace-names,
-- local-names, other-ncnames, et other-uris du type Document
-- (voir le § C.2.5.3)

eNonEmptyOctetStringPrepaddingAdder1 #OCTET-STRING ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByPrepadding
          ENCODED BY eNonEmptyOctetStringWithPrepadding1 }}

-- Insère le prébourrage avant le codage d'un élément des composants
-- attribute-values, content-character-chunks, et other-strings du
-- type Document (voir le § C.2.5.4)

eEncodedCharacterStringPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eEncodedCharacterStringWithPrepadding1 }

-- Insère le prébourrage avant le codage d'un élément des composants
-- element-name-surrogates et attribute-name-surrogates du
-- type Document(voir le § C.2.5.5)

eNameSurrogatePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNameSurrogateWithPrepadding1 }

-- Insère un discriminant avant le codage d'un élément du composant
-- children du type Document (voir le § C.2.11.2 à C.2.11.5)

eDocumentChildDiscriminantAdder1or5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY eDocumentChildWithDiscriminant1or5 }

-- Insère un discriminant avant le codage d'un élément du composant
-- children du type Element(voir le § C.3.7.2 à C.3.7.6)

eElementChildDiscriminantAdder1or5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFiveAlternativeDiscriminant
      ENCODED BY eElementChildWithDiscriminant1or5 }

-- Insère un discriminant avant le codage de la longueur d'une SEQUENCE OF,
-- identifiant une des deux façons de coder la longueur (voir le § C.21)

eNonEmptySequenceOfLengthDiscriminantAdder1 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByTwoAlternativeDiscriminant
      ENCODED BY eNonEmptySequenceOfLengthWithDiscriminant1 }

-- Insère un discriminant avant le codage de la longueur d'une
-- NonEmptyOctetString, identifiant une des trois façons de coder la longueur.
-- Utilisé lorsque le codage débute au second bit d'un octet (voir le § C.22)

eNonEmptyOctetStringLengthDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant2 }

-- Insère un discriminant avant le codage de la longueur d'une
-- NonEmptyOctetString, identifiant une des trois façons de coder la longueur.
-- Utilisé lorsque le codage débute au cinquième bit d'un octet (voir le § C.23)

eNonEmptyOctetStringLengthDiscriminantAdder5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant5 }

-- Insère un discriminant avant le codage de la longueur d'une
-- NonEmptyOctetString, identifiant une des trois façons de coder la longueur.
-- Utilisé lorsque le codage débute au septième bit d'un octet
-- (voir le § C.24)

eNonEmptyOctetStringLengthDiscriminantAdder7 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant7 }
```

```
-- Insère un discriminant avant le codage d'un entier positif,
-- identifiant une des trois façons de le coder. Utilisé lorsque le
-- codage débute au second bit d'un octet (voir le § C.25)

ePositiveIntegerDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant2 }

-- Insère un discriminant avant le codage d'un entier positif,
-- identifiant une des quatre façons de le coder. Utilisé lorsque le
-- codage débute au troisième bit d'un octet (voir le § C.27)

ePositiveIntegerDiscriminantAdder3 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant3 }

-- Insère un discriminant avant le codage d'un entier positif,
-- identifiant une des quatre façons de le coder. Utilisé lorsque le
-- codage débute au quatrième bit d'un octet (voir le § C.28)

ePositiveIntegerDiscriminantAdder4 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant4 }

-- Insère un discriminant avant le codage d'un entier non négatif,
-- identifiant une des trois façons de le coder (voir le § C.26)

eNonNegativeIntegerDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY eNonNegativeIntegerWithDiscriminant2 }

-- Etablit le prébourrage qui a été ajouté avant le composant initial-vocabulary
-- du type Document et code le composant (voir le § C.2.5)

eInitialVocabularyWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 3
              PAD-PATTERN bits:'000'B },
          original {
              ENCODE STRUCTURE {
                  restricted-alphabets {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  encoding-algorithms {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  prefixes {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  namespace-names {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  local-names {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  other-ncnames {
                      ENCODE STRUCTURE {
```

```
                        STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
    }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                other-uris {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
    }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                attribute-values {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                content-character-chunks {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                other-strings {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                element-name-surrogates {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH eRepetitionWithLengthNameSurrogate1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                attribute-name-surrogates {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH eRepetitionWithLengthNameSurrogate1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET }
            WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant chaque élément du composant
-- notations du type Document et code l'élément (voir le § C.2.6.1)

eNotationWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 6
              PAD-PATTERN bits:'110000'B },
          original eNotation7 }
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant chaque élément du composant
-- unparsed-entities du type Document et code l'élément
-- (voir le § C.2.7.1)

eUnparsedEntityWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 7
              PAD-PATTERN bits:'1101000'B },
          original eUnparsedEntity8 }
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant le composant standalone
-- du type Document et code le composant (voir le § C.2.9)

eStandaloneWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 7
              PAD-PATTERN bits:'0000000'B },
          original USE-SET }
```

```
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant chaque élément du composant
-- namespace-attributes du type Element et code l'élément
-- (voir le § C.3.4.2)

eNamespaceAttributeWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 6
              PAD-PATTERN bits:'110011'B
              EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5}
              AS bits:'110011'B },
          original eNamespaceAttribute7
          STRUCTURED WITH {
              ENCODING-SPACE
              EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5} AS bits:'110011'B
      }}
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant chaque élément du composant
-- attributes du type Element et code l'élément(voir le § C.3.6.1)

eAttributeWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 1
              PAD-PATTERN bits:'0'B },
          original eAttribute2 }
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant chaque élément du composant
-- children du type DocumentTypeDeclaration et code l'élément
-- (voir le § C.9.6)

eDocTypeDeclChildWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 8
              PAD-PATTERN bits:'11100001'B },
          original eProcessingInstruction1 }
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant chaque élément des composants
-- restricted-alphabets, encoding-algorithms, prefixes, namespace-names,
-- local-names, other-ncnames, et other-uris du type Document et code l'élément
-- (voir le § C.2.5.3)

eNonEmptyOctetStringWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 1
              PAD-PATTERN bits:'0'B },
          original USE-SET }
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant chaque élément des composants
-- attribute-values, content-character-chunks, et other-strings du
-- type Document et code l'élément (voir le § C.2.5.4)

eEncodedCharacterStringWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 2
              PAD-PATTERN bits:'00'B },
          original USE-SET }
      WITH FastInfosetEncodingSet }


eNameSurrogateWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 6
              PAD-PATTERN bits:'000000'B },
          original eNameSurrogate7 }
```

```
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant le composant
-- literal-qualified-name du type QualifiedNameOrIndex et code le composant.
-- Utilisé lorsque le codage commence au second bit d'un octet (voir le § C.17.3)

eLiteralQualifiedNameWithPrepadding2{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 5
              PAD-PATTERN bits:'11110'B },
          original {
              ENCODE STRUCTURE {
                  prefix eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  namespace-name eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  local-name eIdentifyingStringOrIndex1 }
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
              ENCODING-SPACE
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B }}
      WITH FastInfosetEncodingSet }

-- Etablit le prébourrage qui a été ajouté avant le composant
-- literal-qualified-name du type QualifiedNameOrIndex et code le composant.
-- Utilisé lorsque le codage commence au troisième bit d'un octet
-- (voir le § C.18.3)

eLiteralQualifiedNameWithPrepadding3{<#C>}
#PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 4
              PAD-PATTERN bits:'1111'B
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B },
          original {
              ENCODE STRUCTURE {
                  prefix eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  namespace-name eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  local-name eIdentifyingStringOrIndex1 }
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
              ENCODING-SPACE
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B }}
      WITH FastInfosetEncodingSet }

-- Code le champ longueur qui a été ajouté avant une SEQUENCE OF et code
-- la SEQUENCE OF NonEmptyOctetString (voir le § C.21)

eNonEmptySequenceOfWithLengthNonEmptyOctetString1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eNonEmptyOctetStringPrepaddingAdder1
                  STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }

-- Code le champ longueur qui a été ajouté avant une SEQUENCE OF et code
-- la SEQUENCE OF EncodedCharacterString (voir le § C.21)

eNonEmptySequenceOfWithLengthEncodedCharacterString1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eEncodedCharacterStringDriver1
                  STRUCTURED WITH eRepetitionItems1{<length>}
```

```
            } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }

-- Code le champ longueur qui a été ajouté avant une SEQUENCE OF et code
-- la SEQUENCE OF NameSurrogate (voir le § C.21)

eNonEmptySequenceOfWithLengthNameSurrogate1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eNameSurrogateDriver1
                 STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }

-- Code le champ longueur qui a été ajouté avant une SEQUENCE OF et code
-- la SEQUENCE OF additional-datum (voir le § C.21)

eNonEmptySequenceOfWithLengthAdditionalDatum1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  additional-datum {
                       ENCODE STRUCTURE {
                            id eNonEmptyOctetStringPrepaddingAdder1,
                            data eNonEmptyOctetStringPrepaddingAdder1 }
                       WITH FastInfosetEncodingSet}
                 STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }

-- Code le champ longueur qui a été ajouté avant une NonEmptyOctetString
-- et code la NonEmptyOctetString. Utilisé quand le codage commence au
-- second bit d'un octet (voir le § C.22)

eNonEmptyOctetStringWithLength2{<#C>}
#PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength2,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }

-- Code le champ longueur qui a été ajouté avant une NonEmptyOctetString
-- et code la NonEmptyOctetString. Utilisé quand le codage commence au
-- cinquième bit d'un octet (voir le § C.23)

eNonEmptyOctetStringWithLength5{<#C>}
      #PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength5,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }

-- Code le champ longueur qui a été ajouté avant une NonEmptyOctetString
-- et code la NonEmptyOctetString. Utilisé quand le codage commence au
-- septième bit d'un octet (voir le § C.24)

eNonEmptyOctetStringWithLength7{<#C>}
#PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength7,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }

-- Code le discriminant qui a été ajouté avant un élément du composant children
-- du type Document et code l'élément (voir les § C.2.11.2 à C.2.11.5)

eDocumentChildWithDiscriminant1or5{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
```

```
                ALIGNED TO NEXT octet
                ENCODING-SPACE SIZE 0 },
            discriminant {
                USE #BIT-STRING
                 MAPPING TO BITS {
                    0 TO '0'B,
                    1 TO '11100001'B,
                    2 TO '11100010'B,
                    3 TO '110001'B }
                WITH FastInfosetEncodingSet },
            original {
                ENCODE STRUCTURE {
                    element eElement2,
                    processing-instruction eProcessingInstruction1,
                    comment eComment1,
                    document-type-declaration eDocumentTypeDeclaration7
                    STRUCTURED WITH {
                        ALTERNATIVE DETERMINED BY field-to-be-set
                        USING discriminant }}
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }

    -- Code le discriminant qui a été ajouté avant un élément du composant children
    -- du type Element et code l'élément (voir les § C.3.7.2 à C.3.7.6)

    eElementChildWithDiscriminant1or5{<#C>}
    #PrecededByFiveAlternativeDiscriminant{<#C>} ::= {
        ENCODE STRUCTURE {
            prepadding {
                ALIGNED TO NEXT octet
                ENCODING-SPACE SIZE 0 },
            discriminant {
                USE #BIT-STRING
                MAPPING TO BITS {
                    0 TO '0'B,
                    1 TO '11100001'B,
                    2 TO '110010'B,
                    3 TO '10'B,
                    4 TO '11100010'B }
                WITH FastInfosetEncodingSet },
            original {
                ENCODE STRUCTURE {
                    element eElement2,
                    processing-instruction eProcessingInstruction1,
                    unexpanded-entity-reference eUnexpandedEntityReference7,
                    character-chunk eCharacterChunk3,
                    comment eComment1
                    STRUCTURED WITH {
                        ALTERNATIVE DETERMINED BY field-to-be-set
                        USING discriminant }}
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }

    -- Code le discriminant qui a été ajouté avant la longueur d'une
    -- SEQUENCE OF (identifiant une des deux façons de coder la longueur)
    -- et encode la longueur (voir le § C.21)

    eNonEmptySequenceOfLengthWithDiscriminant1{<#C>}
    #PrecededByTwoAlternativeDiscriminant{<#C>} ::= {
        ENCODE STRUCTURE {
            discriminant {
                USE #BIT-STRING
                MAPPING TO BITS {
                    0 TO '0'B,
                    1 TO '1000'B }
                WITH FastInfosetEncodingSet },
            original {
                ENCODE STRUCTURE {
                    STRUCTURED WITH {
                        ALTERNATIVE DETERMINED BY field-to-be-set
                        USING discriminant }}
                WITH FastInfosetEncodingSet }}
```

```
      WITH FastInfosetEncodingSet }

-- Code le discriminant qui a été ajouté avant la longueur d'une
-- NonEmptyOctetString (identifiant une des trois façons de coder la longueur)
-- et code la longueur. Utilisé quand le codage commence au
-- second bit d'un octet (voir le § C.22)

eNonEmptyOctetStringLengthWithDiscriminant2{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '1000000'B,
                  2 TO '1100000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Code le discriminant qui a été ajouté avant la longueur d'une
-- NonEmptyOctetString (identifiant une des trois façons de coder la longueur)
-- et code la longueur. Utilisé quand le codage commence au cinquième bit
-- cinquième bit d'un octet (voir le § C.23)

eNonEmptyOctetStringLengthWithDiscriminant5{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '1000'B,
                  2 TO '1100'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Code le discriminant qui a été ajouté avant la longueur d'une
-- NonEmptyOctetString (identifiant une des trois façons de coder la longueur)
-- et code la longueur. Utilisé quand le codage commence au
-- septième bit d'un octet (voir le § C.24)

eNonEmptyOctetStringLengthWithDiscriminant7{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '10'B,
                  2 TO '11'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }
```

```
-- Code le discriminant qui a été ajouté avant un entier positif
-- (identifiant une des trois façons de coder l'entier) et code l'entier.
-- Utilisé quand le codage commence au second bit d'un octet
-- (voir le § C.25)

ePositiveIntegerWithDiscriminant2{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '10'B,
                  2 TO '110'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
                  ENCODING-SPACE SIZE self-delimiting-values
                  EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 }
                  AS range:{low 0, high 12}}}  -- Less than '1110'B
      WITH FastInfosetEncodingSet }

-- Code le discriminant qui a été ajouté avant un entier positif
-- (identifiant une des quatre façons de coder l'entier) et code l'entier.
--  Utilisé quand le codage commence au troisième bit d'un octet
-- (voir le § C.27)

ePositiveIntegerWithDiscriminant3{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '100'B,
                  2 TO '101'B,
                  3 TO '110000000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
                  ENCODING-SPACE SIZE self-delimiting-values
                  EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 }
                  AS range:{low 0, high 14}}}  -- Less than '1111'B
      WITH FastInfosetEncodingSet }

-- Code le discriminant qui a été ajouté avant un entier positif
-- (identifiant une des quatre façons de coder l'entier) et code l'entier.
--  Utilisé quand le codage commence au quatrième bit d'un octet
-- (voir le § C.28)

ePositiveIntegerWithDiscriminant4{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '100'B,
                  2 TO '101'B,
                  3 TO '110000000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
```

```
                STRUCTURED WITH {
                    ALTERNATIVE DETERMINED BY field-to-be-set
                    USING discriminant }}
            WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Code le discriminant qui a été ajouté avant un entier non négatif
-- (identifiant une des trois façons de coder l'entier) et code l'entier.
-- (voir le § C.26)

eNonNegativeIntegerWithDiscriminant2{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '1111111'B,
                  1 TO '0'B,
                  2 TO '10'B,
                  3 TO '110'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Code le type Document (voir le § C.2)

eDocument2 #Document ::= {
      ENCODE STRUCTURE {
          additional-data {
              ENCODE STRUCTURE {
                  STRUCTURED WITH eRepetitionWithLengthAdditionalDatum1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING USE-SET,
          initial-vocabulary {
              ENCODE STRUCTURE {
                  STRUCTURED WITH eInitialVocabularyPrepaddingAdder1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING USE-SET,
          notations {
              ENCODE STRUCTURE {
                  eNotationDriver1
                  STRUCTURED WITH eRepetitionWithTerminator8bit1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING USE-SET,
          unparsed-entities {
              ENCODE STRUCTURE {
                  eUnparsedEntityDriver1
                  STRUCTURED WITH eRepetitionWithTerminator8bit1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING USE-SET,
          character-encoding-scheme eNonEmptyOctetStringPrepaddingAdder1
              OPTIONAL-ENCODING USE-SET,
          standalone eStandalonePrepaddingAdder1
              OPTIONAL-ENCODING USE-SET,
          version eNonIdentifyingStringOrIndex1
              OPTIONAL-ENCODING USE-SET,
          children {
              ENCODE STRUCTURE {
                  {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eDocumentChildDiscriminantAdder1or5 }
                      WITH FastInfosetEncodingSet }
                  STRUCTURED WITH eRepetitionWithTerminator4bit1 }
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }
```

```
--Code le type Element (voir le § C.3)

eElement2 #Element ::= {
      ENCODE STRUCTURE {
          namespace-attributes {
              ENCODE STRUCTURE {
                  eNamespaceAttributeDriver1
                  STRUCTURED WITH eRepetitionWithTerminator10bit1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING eNamespaceAttributesOptionality3,
          qualified-name eQualifiedNameOrIndex3,
          attributes {
              ENCODE STRUCTURE {
                  eAttributeDriver1
                  STRUCTURED WITH eRepetitionWithTerminator4bit1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING USE-SET,
          children {
              ENCODE STRUCTURE {
                  {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eElementChildDiscriminantAdder1or5 }
                      WITH FastInfosetEncodingSet }
                  STRUCTURED WITH eRepetitionWithTerminator4bit1 }
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Code le type Attribute (voir le § C.4)

eAttribute2 #Attribute ::= {
      ENCODE STRUCTURE {
          qualified-name eQualifiedNameOrIndex2,
          normalized-value eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }

-- Code le type ProcessingInstruction (voir le § C.5)

eProcessingInstruction1 #ProcessingInstruction ::= {
      ENCODE STRUCTURE {
          target eIdentifyingStringOrIndex1,
          content eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }

-- Code le type UnexpandedEntityReference (voir le § C.6)

eUnexpandedEntityReference7 #UnexpandedEntityReference ::= {
      ENCODE STRUCTURE {
          name eIdentifyingStringOrIndex1,
          system-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET,
          public-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET }
      WITH FastInfosetEncodingSet }

-- Code le type CharacterChunk (voir le § C.7)

eCharacterChunk3 #CharacterChunk ::= {
      ENCODE STRUCTURE {
          character-codes eNonIdentifyingStringOrIndex3 }
      WITH FastInfosetEncodingSet }

-- Code le type Comment (voir le § C.8)

eComment1 #Comment ::= {
      ENCODE STRUCTURE {
          content eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }

-- Code le type DocumentTypeDeclaration (voir le § C.9)

eDocumentTypeDeclaration7 #DocumentTypeDeclaration ::= {
      ENCODE STRUCTURE {
          system-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET,
          public-identifier eIdentifyingStringOrIndex1
```

```
                OPTIONAL-ENCODING USE-SET,
            children {
                ENCODE STRUCTURE {
                    {
                        ENCODE STRUCTURE {
                            STRUCTURED WITH eDocTypeDeclChildPrepaddingAdder1 }
                        WITH FastInfosetEncodingSet }
                    STRUCTURED WITH eRepetitionWithTerminator4bit1 }
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }

-- Code le type UnparsedEntity (voir le § C.10)

eUnparsedEntity8 #UnparsedEntity ::= {
        ENCODE STRUCTURE {
            name eIdentifyingStringOrIndex1,
            system-identifier eIdentifyingStringOrIndex1,
            public-identifier eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET,
            notation-name eIdentifyingStringOrIndex1 }
        WITH FastInfosetEncodingSet }

-- Code le type Notation (voir le § C.11)

eNotation7 #Notation ::= {
        ENCODE STRUCTURE {
            name eIdentifyingStringOrIndex1,
            system-identifier eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET,
            public-identifier eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET }
        WITH FastInfosetEncodingSet }

-- Code le type NamespaceAttribute (voir le § C.12)

eNamespaceAttribute7 #NamespaceAttribute ::= {
        ENCODE STRUCTURE {
            prefix eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET,
            namespace-name eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET }
        WITH FastInfosetEncodingSet }

-- Code le type IdentifyingStringOrIndex (voir le § C.13)

eIdentifyingStringOrIndex1 #IdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string eNonEmptyOctetString2,
            string-index ePositiveInteger2 }
        WITH FastInfosetEncodingSet }

-- Code le type NonIdentifyingStringOrIndex. Utilisé quand le codage commence
-- sur le premier bit d'un octet (voir le § C.14)

eNonIdentifyingStringOrIndex1 #NonIdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string {
                ENCODE STRUCTURE {
                    add-to-table USE-SET,
                    character-string eEncodedCharacterString3 }
                WITH FastInfosetEncodingSet },
            string-index eNonNegativeInteger2 }
        WITH FastInfosetEncodingSet }

-- Code le type NonIdentifyingStringOrIndex. Utilisé quand le codage commence
-- au troisième bit d'un octet (voir le § C.15)

eNonIdentifyingStringOrIndex3 #NonIdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string {
                ENCODE STRUCTURE {
                    add-to-table USE-SET,
                    character-string eEncodedCharacterString5 }
                WITH FastInfosetEncodingSet },
            string-index ePositiveInteger4 }
```

```
      WITH FastInfosetEncodingSet }

-- Code le type NameSurrogate (voir le § C.16)

eNameSurrogate7 #NameSurrogate ::= {
      ENCODE STRUCTURE {
          prefix-string-index ePositiveInteger2
                  OPTIONAL-ENCODING USE-SET,
          namespace-name-string-index ePositiveInteger2
                  OPTIONAL-ENCODING USE-SET,
          local-name-string-index ePositiveInteger2 }
      WITH FastInfosetEncodingSet }

-- Code le type QualifiedNameOrIndex. Utilisé quand le codage commence
-- au second bit d'un octet (voir le § C.17)

eQualifiedNameOrIndex2 #QualifiedNameOrIndex ::= {
      ENCODE STRUCTURE {
          literal-qualified-name {
              ENCODE STRUCTURE {
                  STRUCTURED WITH eLiteralQualifiedNamePrepaddingAdder2 }
              WITH FastInfosetEncodingSet },
          name-surrogate-index ePositiveInteger2
          STRUCTURED WITH eQualifiedNameAlternatives3 }
      WITH FastInfosetEncodingSet }

-- Code le type QualifiedNameOrIndex. Utilisé quand le codage commence
-- au troisième bit d'un octet (voir le § C.18)

eQualifiedNameOrIndex3 #QualifiedNameOrIndex ::= {
      ENCODE STRUCTURE {
          literal-qualified-name {
              ENCODE STRUCTURE {
                  STRUCTURED WITH eLiteralQualifiedNamePrepaddingAdder3 }
              WITH FastInfosetEncodingSet },
          name-surrogate-index ePositiveInteger3
          STRUCTURED WITH eQualifiedNameAlternatives3 }
      WITH FastInfosetEncodingSet }

-- Code le type EncodedCharacterString. Utilisé quand le codage commence
-- au troisième bit d'un octet (voir le § C.19)

eEncodedCharacterString3 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          encoding-format USE-SET,
          octets eNonEmptyOctetString5 }
      WITH FastInfosetEncodingSet }

-- Code le type EncodedCharacterString. Utilisé quand le codage commence
-- au cinquième bit d'un octet (voir le § C.20)

eEncodedCharacterString5 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          encoding-format USE-SET,
          octets eNonEmptyOctetString7 }
      WITH FastInfosetEncodingSet }

-- Code une répétition (SEQUENCE OF NonEmptyOctetString) en insérant un champ
-- longueur avant elle (voir les § C.2.5.3 à C.2.5.5)

eRepetitionWithLengthNonEmptyOctetString1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthNonEmptyOctetString1 }}

-- Code une répétition (SEQUENCE OF EncodedCharacterString) en insérant un champ
-- longueur avant elle (voir les § C.2.5.3 à C.2.5.5)

eRepetitionWithLengthEncodedCharacterString1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthEncodedCharacterString1 }}
```

```
-- Code une répétition (SEQUENCE OF NameSurrogate) en insérant un champ
-- longueur avant elle  (voir les § C.2.5.3 à C.2.5.5)

eRepetitionWithLengthNameSurrogate1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthNameSurrogate1 }}

-- Code une répétition (SEQUENCE OF additional-datum) en insérant un champ
-- longueur avant elle (voir les § C.2.5.3 à C.2.5.5)

eRepetitionWithLengthAdditionalDatum1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthAdditionalDatum1 }}

-- Code le type NonEmptyOctetString. Utilisé quand le codage commence
-- au second bit d'un octet (voir le § C.22)

eNonEmptyOctetString2 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength2 }}

-- Code le type NonEmptyOctetString. Utilisé quand le codage commence
-- au cinquième bit d'un octet (voir le § C.23)

eNonEmptyOctetString5 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength5 }}

-- Code le type NonEmptyOctetString. Utilisé quand le codage commence
-- au septième bit d'un octet (voir le § C.24)

eNonEmptyOctetString7 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength7 }}

-- Code le champ longueur qui a été inséré avant le codage d'une
-- SEQUENCE OF (voir le § C.21)

eNonEmptySequenceOfLength1 #NonEmptySequenceOfLength ::= {
      USE #NonEmptySequenceOfLengthAlternatives1
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptySequenceOfLengthDiscriminantAdder1 }
          WITH FastInfosetEncodingSet }}

-- Code le champ longueur qui a été inséré avant le codage d'une
-- NonEmptyOctetString. Utilisé quand le codage commence au
-- second bit d'un octet (voir le § C.22)

eNonEmptyOctetStringLength2 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}

-- Code le champ longueur qui a été inséré avant le codage d'une
-- NonEmptyOctetString. Utilisé quand le codage commence au
-- cinquième bit d'un octet (voir le § C.23)

eNonEmptyOctetStringLength5 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives5
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder5 }
          WITH FastInfosetEncodingSet }}
```

```
-- Code le champ longueur qui a été inséré avant le codage d'une
-- NonEmptyOctetString. Utilisé quand le codage commence au
-- septième bit d'un octet (voir le § C.24)

eNonEmptyOctetStringLength7 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives7
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder7 }
          WITH FastInfosetEncodingSet }}

-- Code un entier positif. Utilisé quand le codage commence au second bit
-- d'un octet (voir le § C.25)

ePositiveInteger2 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}

-- Code un entier positif. Utilisé quand le codage commence au troisième bit
-- d'un octet (voir le § C.27)

ePositiveInteger3 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives3
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder3 }
          WITH FastInfosetEncodingSet }}

-- Code un entier positif. Utilisé quand le codage commence au quatrième bit
-- d'un octet (voir le § C.28)

ePositiveInteger4 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives4
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder4 }
          WITH FastInfosetEncodingSet }}

-- Code un entier non négatif (voir le § C.26)

eNonNegativeInteger2 #PositiveOrNonNegativeInteger ::= {
      USE #NonNegativeIntegerAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonNegativeIntegerDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}

-- Spécifie comment déterminer la présence du composant namespace-attributes
-- du type Element (voir le § C.3.4.2)

eNamespaceAttributesOptionality3 #OPTIONAL ::= {
      PRESENCE DETERMINED BY handle
      HANDLE "nsa" }

-- Spécifie comment déterminer le remplaçant du type QualifiedNameOrIndex
-- (voir les § C.17.3 et C.18.3)

eQualifiedNameAlternatives3 #ALTERNATIVES ::= {
      ALTERNATIVE DETERMINED BY handle
      HANDLE "qn"
      EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5}
      AS range:{low 0, high 50}}   -- Less than '110011'B

-- Spécifie comment déterminer la terminaison d'une répétition utilisant
-- les 4 bits de terminaison '1111'(voir les § C.2.12, C.3.6.2, C.3.8, et C.9.7)
```

```
eRepetitionWithTerminator4bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'1111'B }}

-- Spécifie comment déterminer la terminaison d'une répétition utilisant
-- les 8 bits de terminaison '11110000'(voir les § C.2.6.2 et C.2.7.2)

eRepetitionWithTerminator8bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'11110000'B }}

-- Spécifie comment déterminer la terminaison d'une répétition utilisant
-- les 10 bits de terminaison '1111000000'(voir le § C.3.4.3)

eRepetitionWithTerminator10bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'1111000000'B
          EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5} AS bits:'110011'B }}

-- Code les éléments d'une SEQUENCE OF, suivant le champ longueur qui a été
-- ajouté (voir les § C.2.5.3 à C.2.5.5)

eRepetitionItems1{<REFERENCE:len>} #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant MULTIPLE OF bit
          DETERMINED BY field-to-be-set USING len }}

-- Code les octets d'une NonEmptyOctetString, suivant le champ longueur qui
-- a été ajouté (voir les § C.22, C.23, et C.24)

eOctetStringOctets1{<REFERENCE:len>} #OCTETS ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant MULTIPLE OF bit
          DETERMINED BY field-to-be-set USING len }}
empty-padding #PAD ::= {
       ENCODING-SPACE SIZE 0
}

FastInfosetEncodingSet #ENCODINGS::= { eDocument2 | empty-padding }
      COMPLETED BY PER-BASIC-UNALIGNED

END

FastInfosetELM
      {joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0)
      modules(0) fast-infoset-elm(2)}
      LINK-DEFINITIONS ::= BEGIN
      IMPORTS FastInfosetEncodingSet, Document FROM FastInfosetEDM;
      ENCODE #Document WITH FastInfosetEncodingSet

END
```

Annexe B

Type de support MIME pour documents Fast Infoset

(La présente annexe fait partie intégrante de la présente Recommandation | Norme internationale)

La présente annexe définit le type de support "application/fastinfoset" qui décrit les documents Fast Infoset.

Le type de support MIME est spécifié ci-dessous en utilisant le gabarit d'enregistrement MIME de l'IETF, et a été enregistré conformément aux procédures de l'IETF.

```
nom du type de support MIME:
      application

nom du sous-type MIME:
      fastinfoset

Paramètres requis:
      aucun.

Paramètres facultatifs:
      aucun.

Considérations sur le codage:
      les infosets XML codés comme documents Fast Infoset vont donner une production de
      données binaires. Ce type de support MIME peut nécessiter un codage ultérieur sur
      les transports incapables de traiter les données binaires.

Considérations sur la sécurité:
      comme les infosets XML codés comme documents Fast Infoset peuvent porter des
      données définies par l'application dont la sémantique est indépendante de tout
      paqueteur MIME (ou du contexte dans lequel le paqueteur MIME est utilisé), on ne
      doit pas s'attendre à être capable de comprendre la sémantique du document Fast
      Infoset sur le seul fondement de la sémantique du paqueteur MIME. Il est donc
      fortement recommandé, chaque fois qu'on utilise le type de support
      "application/fastinfoset", que soient parfaitement comprises, dans ce contexte,
      les implications de l'utilisation du document Fast Infoset sur la sécurité.

Considérations d'interopérabilité:
      il n'y a pas de problèmes connus d'interopérabilité.

Spécification publiée:
      Rec. UIT-T X.891 | ISO/CEI 24824-1.

Applications qui utilisent ce type de support:
      aucune application connue n'utilise ce type de support.

Informations supplémentaires:

      Nombre(s) magique(s):
            un document Fast Infoset peut commencer par une déclaration XML facultative
            qui doit être une des chaînes suivantes, codée en UTF-8:

            <?xml codage='finf'?>
            <?xml codage='finf' standalone='oui'?>
            <?xml codage='finf' standalone='non'?>
            <?xml version='1.0' codage='finf'?>
            <?xml version='1.0' codage='finf' standalone='oui'?>
            <?xml version='1.0' codage='finf' standalone='non'?>
            <?xml version='1.1' codage='finf'?>
            <?xml version='1.1' codage='finf' standalone='oui'?>
            <?xml version='1.1' codage='finf' standalone='non'?>

            Les cinq premiers octets de la déclaration XML codée en UTF-8 sont les
            hexadécimaux 3C 3F 78 6D 6C. Les quatre octets identifiant un document Fast
            Infoset correspondant à la sous-chaîne "finf" codée en UTF-8 sont les
            hexadécimaux 66 69 6E 66.

            Un document Fast Infoset doit commencer par une séquence d'octets des
            hexadécimaux E0 00 00 01 si la déclaration XML facultative est absente.

      Extension(s) de fichier:
            *.finf
```

Adresses personnelle & email à contacter pour des informations complémentaires:
Rapporteur ASN.1 de l'UIT-T (à contacter via tsbmail@itu.int)
Rapporteur ASN.1 de l'ISO/CEI JTC1/SC6 (à contacter via ittfi@iso.org)

Utilisation prévue:
COMMUNE

Auteur/Contrôleur de modifications:
procédures d'adoption conjointes UIT-T | ISO/CEI conformément à la Rec. UIT-T A.23 *Collaboration avec l'Organisation Internationale de Normalisation (ISO) et la Commission électrotechnique internationale (CEI) sur les technologies de l'information*, Annexe A, et Directives ISO/CEI JTC1, Annexe K.

Annexe C

Description du codage d'un document Fast Infoset

(La présente annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

C.1 Document Fast Infoset

C.1.1 La présente annexe décrit de façon informelle (mais précise et complète) les codages qui sont spécifiés au paragraphe 12 et à l'Annexe A. Pour faciliter l'implémentation, toutes les définitions de type ASN.1 du texte normatif sont recopiées dans la présente annexe plutôt que simplement référencées.

C.1.2 Les codages sont décrits en termes d'actions que doit effectuer un codeur, résultant en bits ajoutés à un flux binaire. Les actions que doit effectuer un décodeur ne sont pas décrites explicitement dans la présente annexe, mais peuvent être déduites d'après les actions de codage décrites.

C.1.3 Un document Fast Infoset peut débuter par une déclaration XML (voir le § 12.3) suivie par:

a) les seize bits '1110000000000000' (identification); suivis par

b) les seize bits '0000000000000001' (numéro de version); suivis par

c) le bit '0' (bourrage),

ou par les mêmes trente-trois bits non précédés de déclaration XML. Les trente-trois bits sont immédiatement suivis par le codage d'une valeur abstraite du type **`Document`**, comme décrit au § C.2. Ce codage se termine sur le huitième ou le quatrième bit d'un octet, selon le contenu du document Fast Infoset. Dans le dernier cas, les quatre bits '0000' (bourrage) sont ajoutés au flux binaire.

C.2 Codage du type `Document`

C.2.1 Le type **`Document`** est défini au § 7.2 comme suit:

```
Document ::= SEQUENCE {
      additional-data         SEQUENCE (TAILLE(1..one-meg)) OF
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } FACULTATIF,
      initial-vocabulary      SEQUENCE {
            external-vocabulary         URI FACULTATIF,
            restricted-alphabets        SEQUENCE (TAILLE(1..256)) DE
                  NonEmptyOctetString FACULTATIF,
            encoding-algorithms         SEQUENCE (TAILLE(1..256)) DE
                  NonEmptyOctetString FACULTATIF,
            prefixes                    SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            namespace-names             SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            local-names                 SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            other-ncnames               SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            other-uris                  SEQUENCE (TAILLE(1..one-meg)) DE
                  NonEmptyOctetString FACULTATIF,
            attribute-values            SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            content-character-chunks    SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            other-strings               SEQUENCE (TAILLE(1..one-meg)) DE
                  EncodedCharacterString FACULTATIF,
            element-name-surrogates     SEQUENCE (TAILLE(1..one-meg)) DE
                  NameSurrogate FACULTATIF,
            attribute-name-surrogates   SEQUENCE (TAILLE(1..one-meg)) DE
                  NameSurrogate FACULTATIF }
                  (CONSTRAINED BY {
                        --  Si le composant initial-vocabulary est présent, au moins
                        -- un de ses composants doit être présent -- }) FACULTATIF,
      notations                   SEQUENCE (TAILLE(1..MAX)) DE
                                        Notation FACULTATIF,
```

```
unparsed-entities           SEQUENCE (TAILLE(1..MAX)) DE
                                   UnparsedEntity FACULTATIF,
character-encoding-scheme   NonEmptyOctetString FACULTATIF,
standalone                  BOOLEAN FACULTATIF,
version                     NonIdentifyingStringOrIndex FACULTATIF
                              -- catégorie AUTRE CHAINE --,
children                    SEQUENCE (TAILLE(0..MAX)) DE
      CHOIX {
            element                     Element,
            processing-instruction      ProcessingInstruction,
            comment                     Comment,
            document-type-declaration DocumentTypeDeclaration }}
```

C.2.2 Une valeur du type **Document** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours au second bit d'un octet et se termine au quatrième ou au huitième bit d'un autre octet (qui est le dernier bit de la terminaison '1111' décrite au § C.2.12).

C.2.3 Pour chacun des sept composants facultatifs **additional-data**, **initial-vocabulary**, **notations**, **unparsed-entities**, **character-encoding-scheme**, **standalone** et **version** (dans cet ordre), si le composant est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.2.4 Si le composant facultatif **additional-data** est présent, le nombre de composants de **additional-datum** est codé comme décrit au § C.21, et chacun des composants de **additional-datum** est codé comme décrit dans les deux sous-paragraphes suivants.

C.2.4.1 Le bit '0' (bourrage) est ajouté au flux binaire et le composant **id** est codé comme décrit au § C.22.

C.2.4.2 Le bit '0' (bourrage) est ajouté au flux binaire et le composant **data** est codé comme décrit au § C.22.

C.2.5 Si le composant facultatif **initial-vocabulary** est présent, les trois bits '000' (bourrage) sont ajoutés au flux binaire, et le composant est codé comme décrit dans les cinq sous-paragraphes suivants.

C.2.5.1 Pour chacun des treize composants facultatifs de **initial-vocabulary** (dans l'ordre du texte), si le composant est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.2.5.2 Si le composant facultatif **external-vocabulary** de **initial-vocabulary** est présent, le bit '0' (bourrage) est alors ajouté au flux binaire et le composant est codé comme décrit au § C.22.

C.2.5.3 Pour chacun des composants **restricted-alphabets**, **encoding-algorithms**, **prefixes**, **namespace-names**, **local-names**, **other-ncnames** et **other-uris** (dans cet ordre) présents, le nombre d'éléments **NonEmptyOctetString** dans le composant est codé comme décrit au § C.21, et chaque élément est codé (dans l'ordre) comme suit: le bit '0' (bourrage) est ajouté au flux binaire, et la chaîne **NonEmptyOctetString** est codée comme décrit au § C.22.

C.2.5.4 Pour chaque composant **attribute-values**, **content-character-chunks** et **other-strings** (dans cet ordre) présent, le nombre d'éléments **EncodedCharacterString** dans le composant est codé comme décrit au § C.21, et chaque élément est alors codé (dans l'ordre) comme suit: les deux bits '00' (bourrage) sont ajoutés au flux binaire, et la chaîne **EncodedCharacterString** est codée comme décrit au § C.19.

C.2.5.5 Pour chaque composant **element-name-surrogates** et **attribute-name-surrogates** (dans cet ordre) présent, le nombre d'éléments **NameSurrogate** dans le composant est codé comme décrit au § C.21, et chaque élément est alors codé (dans l'ordre) comme suit: les six bits '000000' (bourrage) sont ajoutés au flux binaire, et le **NameSurrogate** est codé comme décrit au § C.16.

C.2.6 Si le composant facultatif **notations** est présent, il est codé comme décrit aux deux sous-paragraphes suivants.

C.2.6.1 Chaque élément de **notations** (dans l'ordre) est codé comme suit: les six bits '110000' (identification) sont ajoutés au flux binaire, et **Notation** est codé comme décrit au § C.11.

C.2.6.2 Les quatre bits '1111' (terminaison) et les quatre bits '0000' (bourrage) sont ajoutés au flux binaire.

NOTE – Ces bits ne sont pas ajoutés si le composant **notations** est absent.

C.2.7 Si le composant facultatif **unparsed-entities** est présent, il est codé comme décrit aux deux sous-paragraphes suivants.

C.2.7.1 Chaque élément de **unparsed-entities** (dans l'ordre) est codé comme suit: les sept bits '1101000' (identification) sont ajoutés au flux binaire, et **UnparsedEntity** est codé comme décrit au § C.10.

C.2.7.2 Les quatre bits '1111' (terminaison) et les quatre bits '0000' (bourrage) sont ajoutés au flux binaire.

NOTE – Ces bits ne sont pas ajoutés si le composant **unparsed-entities** est absent.

C.2.8 Si le composant facultatif **character-encoding-scheme** est présent, le bit '0' (bourrage) est alors ajouté au flux binaire, et **NonEmptyOctetString** est codé comme décrit au § C.22.

C.2.9 Si le composant facultatif **standalone** est présent, il est codé comme suit: les sept bits '0000000' (bourrage) sont ajoutés au flux binaire. Si la valeur de **standalone** est **VRAI**, le bit '1' est alors ajouté au flux binaire, autrement le bit '0' est ajouté.

C.2.10 Si le composant facultatif **version** est présent, sa valeur est alors codée comme décrit au § C.14.

C.2.11 Si le composant **children** a un ou plusieurs éléments, chaque élément est alors codé (dans l'ordre) comme décrit dans les cinq sous-paragraphes suivants.

C.2.11.1 Le codage de chaque élément doit débuter sur le premier bit d'un octet. Cependant, le dernier bit ajouté peut avoir été soit le huitième soit le quatrième bit d'un octet. S'il était le quatrième bit d'un octet, les bits '0000' (bourrage) sont ajoutés au flux binaire de sorte que le codage de l'élément commence au premier bit du prochain octet.

C.2.11.2 Si l'**element** de remplacement est présent, le bit '0' (identification) est alors ajouté au flux binaire, puis l'**element** est codé comme décrit au § C.3.

C.2.11.3 Si le **processing-instruction** de remplacement est présent, les huit bits '11100001' (identification) sont ajoutés au flux binaire, et **processing-instruction** est codé comme décrit au § C.5.

C.2.11.4 Si le **comment** de remplacement est présent, les huit bits '11100010' (identification) sont alors ajoutés au flux binaire, et **comment** est codé comme décrit au § C.8.

C.2.11.5 Si le **document-type-declaration** de remplacement est présent, les six bits '110001' (identification) sont alors ajoutés au flux binaire, et **document-type-declaration** est codé comme décrit au § C.9.

C.2.12 Les quatre bits '1111' (terminaison) sont ajoutés.

NOTE – Ces bits sont ajoutés même si le composant **children** n'a pas d'éléments.

C.3 Codage du type Element

C.3.1 Le type **Element** est défini au § 7.3 comme suit:

```
Element ::= SEQUENCE {
      namespace-attributes    SEQUENCE (TAILLE(1..MAX)) DE
            NamespaceAttribute FACULTATIF,
      qualified-name          QualifiedNameOrIndex
                                    -- catégorie ELEMENT NAME --,
      attributes              SEQUENCE (TAILLE(1..MAX)) DE
            Attribute OPTIONAL,
      children                SEQUENCE (TAILLE(0..MAX)) DE
            CHOIX {
                  element                       Element,
                  processing-instruction        ProcessingInstruction,
                  unexpanded-entity-reference   UnexpandedEntityReference,
                  character-chunk               CharacterChunk,
                  comment                       Comment }}
```

C.3.2 Une valeur du type **Element** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours au second bit d'un octet et se termine au quatrième ou au huitième bit d'un autre octet (qui est le dernier bit de la terminaison '1111' décrite au § C.3.8).

C.3.3 Si le composant facultatif **attributes** est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.3.4 Si le composant facultatif **namespace-attributes** est présent, il est codé comme décrit aux trois sous-paragraphes suivants.

C.3.4.1 Les quatre bits '1110' (présence) et les deux bits '00' (bourrage) sont ajoutés au flux binaire.

C.3.4.2 Chaque élément de **namespace-attributes** (dans l'ordre) est codé comme suit: les six bits '110011' (identification) sont ajoutés au flux binaire, et **NamespaceAttribute** est codé comme décrit au § C.12.

C.3.4.3 Les quatre bits '1111' (terminaison) et les six bits '000000' (bourrage) sont ajoutés.

NOTE – Ces bits ne sont pas ajoutés si le composant **namespace-attributes** est absent.

C.3.5 La valeur du composant **qualified-name** est codée comme décrit au § C.18.

C.3.6 Si le composant facultatif **attributes** est présent, il est codé comme décrit aux trois sous-paragraphes suivants.

C.3.6.1 Chaque élément de **attributes** (dans l'ordre) est codé comme suit: le bit '0' (identification) est ajouté au flux binaire, et **Attribute** est codé comme décrit au § C.4.

C.3.6.2 Les quatre bits '1111' (terminaison) sont ajoutés.

NOTE – Ces bits ne sont pas ajoutés si le composant **attributes** est absent.

C.3.7 Si le composant **children** a un ou plusieurs éléments, chaque élément est alors codé (dans l'ordre) comme décrit aux six sous-paragraphes suivants.

C.3.7.1 Le codage de chaque élément doit commencer sur le premier bit d'un octet. Cependant, le dernier bit ajouté peut avoir été soit le huitième soit le quatrième bit d'un octet. S'il était le quatrième bit d'un octet, les bits '0000' (bourrage) sont ajoutés de sorte que le codage de l'élément commence sur le premier bit du prochain octet.

C.3.7.2 Si l'**element** de remplacement est présent, le bit '0' (identification) est alors ajouté au flux binaire et l'**element** est codé comme décrit au présent § C.3.

C.3.7.3 Si le **processing-instruction** de remplacement est présent, les huit bits '11100001' (identification) sont alors ajoutés au flux binaire, et **processing-instruction** est codé comme décrit au § C.5.

C.3.7.4 Si la **unexpanded-entity-reference** est présente, les six bits '110010' (identification) sont alors ajoutés au flux binaire, et **unexpanded-entity-reference** est codée comme décrit au § C.6.

C.3.7.5 Si le **character-chunk** de remplacement est présent, les deux bits '10' (identification) sont alors ajoutés au flux binaire, et le **character-chunk** est codé comme décrit au § C.7.

C.3.7.6 Si le **comment** de remplacement est présent, les huit bits '11100010' (identification) sont alors ajoutés au flux binaire, et le **comment** est codé comme décrit au § C.8.

C.3.8 Les quatre bits '1111' (terminaison) sont ajoutés.

NOTE – Ces bits sont ajoutés même si le composant **children** n'a pas d'élément.

C.4 Codage du type Attribute

C.4.1 Le type **Attribute** est défini au § 7.4 comme suit:

```
Attribute ::= SEQUENCE {
      qualified-name      QualifiedNameOrIndex
                                -- catégorie ATTRIBUTE NAME --,
      normalized-value    NonIdentifyingStringOrIndex
                                -- catégorie ATTRIBUTE VALUE -- }
```

C.4.2 Une valeur du type **Attribute** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours au second bit d'un octet et se termine sur le huitième bit d'un autre octet.

C.4.3 La valeur de **qualified-name** est codée comme décrit au § C.17.

C.4.4 La valeur de **normalized-value** est codée comme décrit au § C.14.

C.5 Codage du type ProcessingInstruction

C.5.1 Le type **ProcessingInstruction** est défini au § 7.5 comme suit:

```
ProcessingInstruction ::= SEQUENCE {
      target        IdentifyingStringOrIndex
                          -- catégorie OTHER NCNAME --,
      content       NonIdentifyingStringOrIndex
                          -- catégorie OTHER STRING -- }
```

C.5.2 Une valeur du type **ProcessingInstruction** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours sur le premier bit d'un octet et se termine au huitième bit d'un autre octet.

C.5.3 La valeur de **target** est codée comme décrit au § C.13.

C.5.4 La valeur de **content** est codée comme décrit au § C.14.

C.6 Codage du type `UnexpandedEntityReference`

C.6.1 Le type **`UnexpandedEntityReference`** est défini au § 7.6 comme suit:

```
UnexpandedEntityReference ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                  -- catégorie OTHER NCNAME --,
      system-identifier     IdentifyingStringOrIndex FACULTATIF
                                  -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                  -- catégorie OTHER URI -- }
```

C.6.2 Une valeur du type **`UnexpandedEntityReference`** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours sur le septième bit d'un octet et se termine sur le huitième bit d'un autre octet.

C.6.3 Pour chacun des composants facultatifs **`system-identifier`** et **`public-identifier`** (dans cet ordre), si le composant est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.6.4 La valeur de **`name`** est codée comme décrit au § C.13.

C.6.5 Si le composant facultatif **`system-identifier`** est présent, il est codé comme décrit au § C.13.

C.6.6 Si le composant facultatif **`public-identifier`** est présent, il est codé comme décrit au § C.13.

C.7 Codage du type `CharacterChunk`

C.7.1 Le type **`CharacterChunk`** est défini au § 7.7 comme suit:

```
CharacterChunk ::= SEQUENCE {
      character-codes             NonIdentifyingStringOrIndex
                                        -- catégorie CONTENT CHARACTER CHUNK -- }
```

C.7.2 Une valeur du type **`CharacterChunk`** type est codée en effectuant l'action suivante.

NOTE – Un codage de ce type débute toujours au troisième bit d'un octet et se termine sur le huitième bit du même octet ou d'un autre.

C.7.3 La valeur de **`character-codes`** est codée comme décrit au § C.15.

C.8 Codage du type `Comment`

C.8.1 Le type **`Comment`** est défini au § 7.8 comme suit:

```
Comment ::= SEQUENCE {
      content     NonIdentifyingStringOrIndex -- catégorie OTHER STRING --}
```

C.8.2 Une valeur du type **`Comment`** type est codée en effectuant l'action suivante.

NOTE – Un codage de ce type débute toujours sur le premier bit d'un octet et se termine sur le huitième bit du même octet ou d'un autre.

C.8.3 La valeur de **`content`** est codée comme décrit au § C.14.

C.9 Codage du type `DocumentTypeDeclaration`

C.9.1 Le type **`DocumentTypeDeclaration`** est défini au § 7.9 comme suit:

```
DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier     IdentifyingStringOrIndex FACULTATIF
                                  -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                  -- catégorie OTHER URI --,
      children              SEQUENCE (TAILLE(0..MAX)) DE
            ProcessingInstruction }
```

C.9.2 Une valeur du type **`DocumentTypeDeclaration`** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours sur le septième bit d'un octet et se termine sur le quatrième bit d'un autre octet (qui est le dernier bit de la terminaison '1111' décrite au § C.9.7).

C.9.3 Pour chacun des composants facultatifs **`system-identifier`** et **`public-identifier`** (dans cet ordre), si le composant est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.9.4 Si le composant facultatif **system-identifier** est présent, il est codé comme décrit au § C.13.

C.9.5 Si le composant facultatif **public-identifier** est présent, il est codé comme décrit au § C.13.

C.9.6 Si le composant **children** a un ou plusieurs éléments, chaque élément est alors codé comme suit: les huit bits '11100001' (identification) sont ajoutés au flux binaire, et **ProcessingInstruction** est codé comme décrit au § C.5.

C.9.7 Les quatre bits '1111' (terminaison) sont ajoutés.

NOTE – Ces bits sont ajoutés même si le composant **children** n'a pas d'élément.

C.10 Codage du type UnparsedEntity

C.10.1 Le type **UnparsedEntity** est défini au § 7.10 comme suit:

```
UnparsedEntity ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                -- catégorie OTHER NCNAME --,
      system-identifier     IdentifyingStringOrIndex
                                -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                -- catégorie OTHER URI --,
      notation-name         IdentifyingStringOrIndex
                                -- catégorie OTHER NCNAME -- }
```

C.10.2 Une valeur du type **UnparsedEntity** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours au huitième bit d'un octet et se termine au huitième bit d'un autre octet.

C.10.3 Si le composant facultatif **public-identifier** est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.10.4 La valeur de **name** est codée comme décrit au § C.13.

C.10.5 La valeur de **system-identifier** est codée comme décrit au § C.13.

C.10.6 Si le composant facultatif **public-identifier** est présent, il est codé comme décrit au § C.13.

C.10.7 La valeur de **name** est codée comme décrit au § C.13.

C.11 Codage du type Notation

C.11.1 Le type **Notation** est défini au § 7.11 comme suit:

```
Notation ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                -- catégorie OTHER NCNAME --,
      system-identifier     IdentifyingStringOrIndex FACULTATIF
                                -- catégorie OTHER URI --,
      public-identifier     IdentifyingStringOrIndex FACULTATIF
                                -- catégorie OTHER URI -- }
```

C.11.2 Une valeur du type **Notation** type est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours au septième bit d'un octet et se termine au huitième bit d'un autre octet.

C.11.3 Pour chacun des composants facultatifs **system-identifier** et **public-identifier** (dans cet ordre), si le composant est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.11.4 La valeur de **name** est codée comme décrit au § C.13.

C.11.5 Si le composant facultatif **system-identifier** est présent, il est codé comme décrit au § C.13.

C.11.6 Si le composant facultatif **public-identifier** est présent, il est codé comme décrit au § C.13.

C.12 Codage du type NamespaceAttribute

C.12.1 Le type **NamespaceAttribute** est défini au § 7.12 comme suit:

```
NamespaceAttribute ::= SEQUENCE {
      prefix              IdentifyingStringOrIndex FACULTATIF
                                -- catégorie PREFIX --,
      namespace-name      IdentifyingStringOrIndex FACULTATIF
                                -- catégorie NAMESPACE NAME -- }
```

C.12.2 Une valeur du type **NamespaceAttribute** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours au huitième bit d'un octet et se termine au huitième bit d'un autre octet.

C.12.3 Si le composant facultatif **prefix** est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.12.4 Si le composant facultatif **namespace-name** est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.12.5 Si le composant facultatif **prefix** est présent, il est codé comme décrit au § C.13.

C.12.6 Si le composant facultatif **namespace-name** est présent, il est codé comme décrit au § C.13.

C.13 Codage du type IdentifyingStringOrIndex

C.13.1 Le type **IdentifyingStringOrIndex** est défini au § 7.13 comme suit:

```
IdentifyingStringOrIndex ::= CHOIX {
      literal-character-string   NonEmptyOctetString,
      string-index               ENTIER (1..one-meg) }
```

C.13.2 Une valeur du type **IdentifyingStringOrIndex** est codée en effectuant les actions suivantes.

NOTE – Un codage de ce type débute toujours au premier bit d'un octet et se termine au huitième bit du même octet ou d'un autre.

C.13.3 Si la chaîne de remplacement **literal-character-string** est présente, le bit '0' (discriminant) est alors ajouté au flux binaire, et la chaîne **literal-character-string** est codée comme décrit au § C.22.

C.13.4 Si l'indice de remplacement **string-index** est présent, le bit '1' (discriminant) est alors ajouté au flux binaire, et **string-index** est codé comme décrit au § C.25.

C.14 Codage du type NonIdentifyingStringOrIndex commençant sur le premier bit d'un octet

C.14.1 Le type **NonIdentifyingStringOrIndex** est défini au § 7.14 comme suit:

```
NonIdentifyingStringOrIndex ::= CHOIX {
      literal-character-string    SEQUENCE {
            add-to-table                BOOLEEN,
            character-string            EncodedCharacterString },
      string-index                ENTIER (0..one-meg) }
```

C.14.2 Ce paragraphe C.14 est invoqué pour coder une valeur du type **NonIdentifyingStringOrIndex** lorsque le codage doit commencer au premier bit d'un octet (voir aussi le § C.15). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit du même octet ou d'un autre.

C.14.3 Si la chaîne de remplacement **literal-character-string** est présente, le bit '0' (discriminant) est alors ajouté au flux binaire, et **literal-character-string** est codé comme décrit aux deux sous-paragraphes suivants.

C.14.3.1 Si la valeur du composant **add-to-table** est **VRAI**, le bit '1' est alors ajouté au flux binaire, autrement le bit '0' est ajouté.

C.14.3.2 La valeur du composant **character-string** est codée comme décrit au § C.19.

C.14.4 Si l'indice de remplacement **string-index** est présent, le bit '1' (discriminant) est alors ajouté au flux binaire, et **string-index** est codé comme décrit au § C.26.

C.15 Codage du type `NonIdentifyingStringOrIndex` commençant au troisième bit d'un octet

C.15.1 Le type **`NonIdentifyingStringOrIndex`** est défini au § 7.14 comme suit:

```
NonIdentifyingStringOrIndex ::= CHOIX {
      literal-character-string    SEQUENCE {
            add-to-table              BOOLEEN,
            character-string          EncodedCharacterString },
      string-index                ENTIER (0..one-meg) }
```

C.15.2 Ce paragraphe C.15 est invoqué pour coder une valeur du type **`NonIdentifyingStringOrIndex`** lorsque ce codage doit commencer sur le troisième bit d'un octet (voir aussi le § C.14). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit du même ou d'un autre octet.

C.15.3 Si la chaîne de remplacement **`literal-character-string`** est présente, le bit '0' (discriminant) est alors ajouté au flux binaire, et **`literal-character-string`** est codée comme décrit aux deux sous-paragraphes suivants.

C.15.3.1 Si la valeur du composant **`add-to-table`** est **`VRAI`**, le bit '1' est alors ajouté au flux binaire, autrement le bit '0' est ajouté.

C.15.3.2 La valeur du composant **`character-string`** est codée comme décrit au § C.20.

C.15.4 Si l'indice de remplacement **`string-index`** est présent, le bit '1' (discriminant) est alors ajouté au flux binaire, et **`string-index`** est codé comme décrit au § C.28.

C.16 Codage du type `NameSurrogate`

C.16.1 Le type **`NameSurrogate`** est défini au § 7.15 comme suit:

```
NameSurrogate ::= SEQUENCE {
      prefix-string-index           ENTIER(1..one-meg) FACULTATIF,
      namespace-name-string-index   ENTIER(1..one-meg) FACULTATIF,
      local-name-string-index       ENTIER(1..one-meg) }
      (CONTRAINT PAR   {-- prefix-string-index ne doit être présent que si
                       -- namespace-name-string-index est présent --})
```

C.16.2 Une valeur du type **`NameSurrogate`** est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type débute toujours sur le septième bit d'un octet et se termine sur le huitième bit d'un autre octet.

C.16.3 Si le composant facultatif **`prefix-string-index`** est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.16.4 Si le composant facultatif **`namespace-name-string-index`** est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.16.5 Si le composant facultatif **`prefix-string-index`** est présent, le bit '0' (bourrage) est alors ajouté au flux binaire, et le composant est codé comme décrit au § C.25.

C.16.6 Si le composant facultatif **`namespace-name-string-index`** est présent, le bit '0' (bourrage) est alors ajouté au flux binaire, et le composant est codé comme décrit au § C.25.

C.16.7 Le bit '0' (bourrage) est ajouté au flux binaire, et le composant **`local-name-string-index`** est codé comme décrit au § C.25.

C.17 Codage du type `QualifiedNameOrIndex` commençant sur le second bit d'un octet

C.17.1 Le type **`QualifiedNameOrIndex`** est défini au § 7.16 comme suit:

```
QualifiedNameOrIndex ::= CHOIX {
      literal-qualified-name  SEQUENCE {
            prefix                IdentifyingStringOrIndex FACULTATIF
                                        -- catégorie PREFIX --,
            namespace-name        IdentifyingStringOrIndex FACULTATIF
                                        -- catégorie NAMESPACE NAME --,
            local-name            IdentifyingStringOrIndex
                                        -- catégorie LOCAL NAME -- },
      name-surrogate-index    ENTIER (1..one-meg) }
```

C.17.2 Ce paragraphe C.17 est invoqué pour coder une valeur du type **QualifiedNameOrIndex** quand le codage doit commencer sur le second bit d'un octet (voir aussi le § C.18). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit du même ou d'un autre octet.

C.17.3 Si le nom de remplacement **literal-qualified-name** est présent, les quatre bits '1111' (identification) et le bit '0' (bourrage) sont ajoutés au flux binaire, et **literal-qualified-name** est codé comme décrit aux quatre sous-paragraphes suivants.

C.17.3.1 Pour chacun des composants facultatifs **prefix** et **namespace-name** (dans cet ordre), si le composant est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.17.3.2 Si le composant facultatif **prefix** est présent, il est codé comme décrit au § C.13.

C.17.3.3 Si le composant facultatif **namespace-name** est présent, il est codé comme décrit au § C.13.

C.17.3.4 Le composant **local-name** est codé comme décrit au § C.13.

C.17.4 Si l'indice de remplacement **name-surrogate-index** est présent, il est codé comme décrit au § C.25.

C.18 Codage du type QualifiedNameOrIndex commençant sur le troisième bit d'un octet

C.18.1 Le type **QualifiedNameOrIndex** est défini au § 7.16 comme suit:

```
QualifiedNameOrIndex ::= CHOIX {
      literal-qualified-name  SEQUENCE {
            prefix                IdentifyingStringOrIndex FACULTATIF
                                        -- catégorie PREFIX --,
            namespace-name        IdentifyingStringOrIndex FACULTATIF
                                        -- catégorie NAMESPACE NAME --,
            local-name            IdentifyingStringOrIndex
                                        -- catégorie LOCAL NAME -- },
      name-surrogate-index    ENTIER (1..one-meg) }
```

C.18.2 Ce paragraphe C.18 est invoqué pour coder une valeur du type **QualifiedNameOrIndex** quand le codage doit commencer sur le troisième bit d'un octet (voir aussi le § C.17). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit du même ou d'un autre octet.

C.18.3 Si le nom de remplacement **literal-qualified-name** est présent, les quatre bits '1111' (identification) sont ajoutés au flux binaire, et **literal-qualified-name** est codé comme décrit aux quatre sous-paragraphes suivants.

C.18.3.1 Pour chacun des composants facultatifs **prefix** et **namespace-name** (dans cet ordre), si le composant est présent, le bit '1' (présence) est alors ajouté au flux binaire, autrement le bit '0' (absence) est ajouté.

C.18.3.2 Si le composant facultatif **prefix** est présent, il est codé comme décrit au § C.13.

C.18.3.3 Si le composant facultatif **namespace-name** est présent, il est codé comme décrit au § C.13.

C.18.3.4 Le composant **local-name** est codé comme décrit au § C.13.

C.18.4 Si l'indice de remplacement **name-surrogate-index** est présent, il est codé comme décrit au § C.27.

C.19 Codage du type EncodedCharacterString commençant sur le troisième bit d'un octet

C.19.1 Le type **EncodedCharacterString** est défini au § 7.17 comme suit:

```
EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOIX {
            utf-8                 NULL,
            utf-16                NULL,
            restricted-alphabet   ENTIER(1..256),
            encoding-algorithm    ENTIER(1..256) },
      octets                NonEmptyOctetString }
```

C.19.2 Ce paragraphe C.19 est invoqué pour coder une valeur du type **EncodedCharacterString** quand le codage doit commencer sur le troisième bit d'un octet (voir aussi le § C.20). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit d'un autre octet.

C.19.3 La valeur du composant **encoding-format** est codée comme décrit aux quatre sous-paragraphes suivants.

C.19.3.1 Si l'**utf-8** de remplacement est présent, les deux bits '00' (discriminant) sont alors ajoutés au flux binaire.

C.19.3.2 Si l'**utf-16** de remplacement est présent, les deux bits '01' (discriminant) sont alors ajoutés au flux binaire.

C.19.3.3 Si l'alphabet de remplacement **restricted-alphabet** est présent, les deux bits '10' (discriminant) sont alors ajoutés au flux binaire, et **restricted-alphabet** est codé comme décrit au § C.29.

C.19.3.4 Si l'algorithme de remplacement **encoding-algorithm** est présent, les deux bits '11' (discriminant) sont alors ajoutés au flux binaire, et **encoding-algorithm** est codé comme décrit au § C.29.

C.19.4 Le composant **octets** est codé comme décrit au § C.23.

C.20 Codage du type EncodedCharacterString commençant sur le cinquième bit d'un octet

C.20.1 Le type **EncodedCharacterString** est défini au § 7.17 comme suit:

```
EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOICE {
            utf-8                   NULL,
            utf-16                  NULL,
            restricted-alphabet     ENTIER(1..256),
            encoding-algorithm      ENTIER(1..256) },
      octets                NonEmptyOctetString }
```

C.20.2 Ce paragraphe C.20 est invoqué pour coder une valeur du type **EncodedCharacterString** quand le codage doit commencer sur le cinquième bit d'un octet (voir aussi le § C.19). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit d'un autre octet.

C.20.3 La valeur du composant **encoding-format** est codée comme décrit aux quatre sous-paragraphes suivants.

C.20.3.1 Si l'**utf-8** de remplacement est présent, les deux bits '00' (discriminant) sont alors ajoutés au flux binaire.

C.20.3.2 Si l'**utf-16** de remplacement est présent, les deux bits '01' (discriminant) sont alors ajoutés au flux binaire.

C.20.3.3 Si l'algorithme de remplacement **restricted-alphabet** est présent, les deux bits '10' (discriminant) sont alors ajoutés au flux binaire, et **restricted-alphabet** est codé comme décrit au § C.29.

C.20.3.4 Si l'algorithme de remplacement **encoding-algorithm** est présent, les deux bits '11' (discriminant) sont alors ajoutés au flux binaire, et **encoding-algorithm** est codé comme décrit au § C.29.

C.20.4 Le composant **octets** est codé comme décrit au § C.24.

C.21 Codage de la longueur d'un type sequence-of

C.21.1 Le présent paragraphe est invoqué pour coder la longueur d'un type sequence-of qui est codé avec un champ longueur précédant les éléments du type sequence-of.

NOTE – Ce codage commence toujours sur le premier bit d'un octet et se termine sur le huitième bit du même ou d'un autre octet.

C.21.2 Si la valeur est dans la gamme 1 à 128, le bit '0' est alors ajouté au flux binaire, et la valeur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de sept bits et est ajoutée.

C.21.3 Si la valeur est dans la gamme 129 à 2^{20}, le bit '1' et les trois bits '000' (bourrage) sont ajoutés au flux binaire, et la valeur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de vingt bits et est ajoutée.

C.22 Codage du type NonEmptyOctetString commençant sur le second bit d'un octet

C.22.1 Le type **NonEmptyOctetString** est défini au § 7.2 comme suit:

```
NonEmptyOctetString ::= OCTET STRING (TAILLE(1..four-gig))
```

C.22.2 Ce paragraphe C.22 est invoqué pour coder une valeur du type **NonEmptyOctetString** quand le codage doit commencer sur le second bit d'un octet (voir aussi le § C.23 et C.24). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit d'un autre octet.

C.22.3 La longueur de la chaîne d'octets est codée comme décrit aux trois sous-paragraphes suivants.

C.22.3.1 Si la longueur est dans la gamme de 1 à 64, le bit '0' est alors ajouté au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de six bits et est ajoutée.

C.22.3.2 Si la longueur est dans la gamme de 65 à 320, les deux bits '10' et les cinq bits '00000' (bourrage) sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de huit bits et est ajoutée.

C.22.3.3 Si la longueur est dans la gamme de 321 à 2^{32}, les deux bits '11' et les cinq bits '00000' (bourrage) sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de trente-deux bits et est ajoutée.

C.22.4 Les bits formant les octets de la chaîne d'octets sont ajoutés au flux binaire (dans l'ordre).

C.23 Codage de la chaîne `NonEmptyOctetString` commençant sur le cinquième bit d'un octet

C.23.1 Le type **`NonEmptyOctetString`** est défini au § 7.2 comme suit:

```
NonEmptyOctetString ::= OCTET STRING (TAILLE(1..four-gig))
```

C.23.2 Ce paragraphe C.23 est invoqué pour coder une valeur du type **`NonEmptyOctetString`** quand le codage doit commencer sur le cinquième bit d'un octet (voir aussi les § C.22 et C.24). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit d'un autre octet.

C.23.3 La longueur de la chaîne d'octets est codée comme décrit aux trois sous-paragraphes suivants.

C.23.3.1 Si la longueur est dans la gamme de 1 à 8, le bit '0' est alors ajouté au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de trois bits et est ajoutée.

C.23.3.2 Si la longueur est dans la gamme de 9 à 264, les deux bits '10' et les deux bits '00' (bourrage) sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de huit bits et est ajoutée.

C.23.3.3 Si la longueur est dans la gamme de 265 à 2^{32}, les deux bits '11' et les deux bits '00' (bourrage) sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de trente-deux bits et est ajoutée.

C.23.4 Les bits formant les octets de la chaîne d'octets sont ajoutés au flux binaire (dans l'ordre).

C.24 Codage du type `NonEmptyOctetString` commençant sur le septième bit d'un octet

C.24.1 Le type **`NonEmptyOctetString`** est défini au § 7.2 comme suit:

```
NonEmptyOctetString ::= OCTET STRING (TAILLE(1..four-gig))
```

C.24.2 Ce paragraphe C.24 est invoqué pour coder une valeur du type **`NonEmptyOctetString`** quand le codage doit commencer sur le septième bit d'un octet (voir aussi les § C.22 et C.23). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit d'un autre octet.

C.24.3 La longueur de la chaîne d'octets est codée comme décrit aux trois sous-paragraphes suivants.

C.24.3.1 Si la longueur est dans la gamme 1 à 2, le bit '0' est alors ajouté au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de un bit et est ajoutée.

C.24.3.2 Si la longueur est dans la gamme de 3 à 258, les deux bits '10' sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de huit bits et est ajoutée.

C.24.3.3 Si la longueur est dans la gamme de 259 à 2^{32}, les deux bits '11' sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de trente-deux bits et est ajoutée.

C.24.4 Les bits formant les octets de la chaîne d'octets sont ajoutés au flux binaire (dans l'ordre).

C.25 Codage des entiers dans la gamme 1 à 2^{20} commençant sur le second bit d'un octet

C.25.1 Ce paragraphe C.25 est invoqué pour coder une valeur entière dans la gamme 1 à 2^{20} quand le codage doit commencer sur le second bit d'un octet (voir aussi les § C.26, C.27 et C.28). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit du même ou d'un autre octet.

C.25.2 Si la valeur est dans la gamme 1 à 64, le bit '0' est alors ajouté au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de six bits et est ajoutée.

C.25.3 Si la valeur est dans la gamme 65 à 8256, les deux bits '10' sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de treize bits et est ajoutée.

C.25.4 Si la valeur est dans la gamme 8257 à 2^{20}, les deux bits '11' et le bit '0' (bourrage) sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de vingt bits et est ajoutée.

C.26 Codage des entiers dans la gamme 0 à 2^{20} commençant sur le second bit d'un octet

C.26.1 Ce paragraphe C.26 est invoqué pour coder une valeur entière dans la gamme 0 à 2^{20} quand le codage doit commencer sur le second bit d'un octet (voir aussi les § C.25, C.27 et C.28). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit du même ou d'un autre octet.

C.26.2 Si la valeur est zéro, les sept bits '1111111' sont alors ajoutés au flux binaire. Autrement, la valeur est codée comme décrit au § C.25.

C.27 Codage des entiers dans la gamme 1 à 2^{20} commençant sur le troisième bit d'un octet

C.27.1 Ce paragraphe C.27 est invoqué pour coder une valeur entière dans la gamme 1 à 2^{20} quand le codage doit commencer sur le troisième bit d'un octet (voir aussi les § C.25, C.26 et C.28). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit du même ou d'un autre octet.

C.27.2 Si la valeur est dans la gamme 1 à 32, le bit '0' est alors ajouté au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de cinq bits et est ajoutée.

C.27.3 Si la valeur est dans la gamme 33 à 2080, les trois bits '100' sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de onze bits et est ajoutée.

C.27.4 Si la valeur est dans la gamme 2081 à 526368, les trois bits '101' sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de dix-neuf bits et est ajoutée.

C.27.5 Si la valeur est dans la gamme 526369 à 2^{20}, les trois bits '110' et les sept bits '0000000' (bourrage) sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de vingt bits et est ajoutée.

C.28 Codage des entiers dans la gamme 1 à 2^{20} commençant sur le quatrième bit d'un octet

C.28.1 Ce paragraphe C.28 est invoqué pour coder une valeur entière dans la gamme 1 à 2^{20} quand le codage doit commencer sur le quatrième bit d'un octet (voir aussi les § C.25, C.26 et C.27). La valeur est codée en effectuant les actions suivantes (dans l'ordre).

NOTE – Un codage de ce type se termine toujours sur le huitième bit du même ou d'un autre octet.

C.28.2 Si la valeur est dans la gamme 1 à 16, le bit '0' est alors ajouté au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de quatre bits et est ajoutée.

C.28.3 Si la valeur est dans la gamme 17 à 1040, les trois bits '100' sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de dix bits et est ajoutée.

C.28.4 Si la valeur est dans la gamme 1041 à 263184, les trois bits '101' sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de dix-huit bits et est ajoutée.

C.28.5 Si la valeur est dans la gamme 263185 à 2^{20}, les trois bits '110' et les six bits '000000' (bourrage) sont ajoutés au flux binaire, et la longueur, moins la limite inférieure de la gamme, est codée comme un entier non signé dans un champ de vingt bits et est ajoutée.

C.29 Codage des entiers dans la gamme 1 à 256

C.29.1 Ce paragraphe C.29 est invoqué pour coder une valeur entière dans la gamme 1 à 256.

NOTE – Un codage de ce type débute toujours soit sur le cinquième bit soit sur le septième bit d'un octet et se termine (respectivement) sur le quatrième bit ou le sixième bit de l'octet suivant.

C.29.2 La valeur, moins la limite inférieur de la gamme, est codée comme un entier non signé dans un champ de huit bits et est ajoutée au flux binaire.

Annexe D

Exemples de codage d'infosets XML comme documents Fast Infoset

(La présente annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

D.1 Introduction des exemples

D.1.1 La présente annexe utilise les conventions typographiques suivantes pour les nombres:

a) pour un nombre représenté en base dix, on utilise le **`Courier gras`** pour les chiffres du nombre, suivis de l'indice "10" (par exemple, $\mathbf{11}_{10}$);

b) pour un nombre représenté en base seize (un nombre hexadécimal), on utilise le **`Courier gras`** pour les chiffres du nombre, suivis de l'indice "16" (par exemple, $\mathbf{0b1f}_{16}$); et

c) si la base d'un nombre est énoncée explicitement, l'indice est alors omis.

D.1.2 La présente annexe présente deux exemples de codages possibles d'un ordre du langage d'affaires universel (UBL, *universal business language*) [1] dans un document Fast Infoset. UBL est conçu pour fournir une syntaxe commerciale universellement comprise et reconnue pour des documents d'affaire officiels.

D.1.3 L'infoset XML pour l'exemple d'ordre UBL est présenté au § D.3.

D.1.4 Le premier document Fast Infoset a un vocabulaire initial qui fait référence à un vocabulaire externe. Le § D.4 décrit le contenu du vocabulaire externe, les octets du document Fast Infoset, et explique quelques séquences d'octets.

D.1.5 Le second Fast Infoset n'a pas de vocabulaire initial. Le paragraphe D.5 décrit les octets de ce document Fast Infoset, et donne l'explication de certaines séquences d'octets.

NOTE – Le vocabulaire final de ce document Fast Infoset est le même que le vocabulaire final du document Fast Infoset décrit au § D.4.

D.1.6 Les octets des § D.4 et D.5 sont présentés dans une série de tableaux à deux colonnes. La première colonne donne la liste des positions de départ en hexadécimal de 32 octets consécutifs du document Fast Infoset, et la seconde colonne donne la liste des octets en notation hexadécimale. Les caractères hexadécimaux qui contiennent des bits correspondant à l'identification et la terminaison des éléments d'information sont soulignés.

D.1.7 Les explications de certaines séquences d'octets des documents Fast Infoset (en D.4 et D.5) sont présentées dans des tableaux avec les colonnes suivantes:

a) La colonne 1 présente la position, en hexadécimal, du ou des octets listés dans la colonne 2.

b) La colonne 2 présente le ou les octets du document Fast Infoset associé à l'élément d'information pertinent et les propriétés de l'élément. Un octet est représenté en base deux suivi par le même octet représenté en base seize (hexadécimal) entre parenthèses, par exemple, **`11110000 (f0)`**.

c) La colonne 3 présente, en détail, une description des octets de la colonne 2, et se réfère aux paragraphes de l'Annexe C pour d'autres explications et éclaircissements.

d) La colonne 4 présente une portion de l'infoset XML ou une portion du document XML 1.0 (si applicable) correspondant aux octets de la colonne 2.

D.1.8 Dans ces exemples tous les tronçons d'éléments d'information **character** qui contiennent moins de 6 caractères sont ajoutés au tableau CONTENT CHARACTER CHUNK, et la propriété **[normalized value]** de tous les éléments d'information **attribute** qui contiennent moins de 6 caractères sont ajoutés au tableau ATTRIBUTE VALUE.

D.1.9 Les tailles du document XML 1.0 et des documents Fast Infoset, et les tailles compressées (utilisant GZIP) de ces documents dont données au § D.2.

D.2 Taille des documents exemples (y compris la compression fondée sur la redondance)

D.2.1 Le Tableau D.1 présente la taille de tous les documents. La colonne 1 donne la liste des documents UBL, la colonne 2 donne la liste des tailles des documents, et la colonne 3 donne la liste des tailles de documents compressés par GZIP (avec les options par défaut) [2].

NOTE 1 – Le document XML 1.0 en ordre UBL ne contient pas d'espaces blancs (voir le § D.3.1.2).

NOTE 2 – Pour chaque document, tous les caractères sont codés en utilisant le codage de caractères UTF-8.

NOTE 3 – Il n'y a pas de déclaration XML (voir le § 12.3) pour les documents Fast Infoset.

Tableau D.1/X.891 – Tailles initiales et compressées par GZIP des documents

Document UBL	Taille	Taille compressée par GZIP
Document XML 1.0	3311	893
Document Fast Infoset avec vocabulaire externe	684	546
Document Fast Infoset sans vocabulaire initial	1322	860

D.2.2 La taille du document Fast Infoset avec référence à un vocabulaire externe est la plus faible, et elle l'est également en compression GZIP. Le ratio de la taille compressée GZIP par rapport à la taille du document Fast Infoset implique que ce document Fast Infoset a peu d'information redondante.

D.2.3 Dans tous les cas, les tailles des documents Fast Infoset compressés par GZIP sont plus faibles que la taille compressée par GZIP du document XML 1.0. De plus, la taille du document Fast Infoset faisant référence à un vocabulaire externe est plus faible que la taille compressée GZIP du document XML 1.0.

D.3 Exemple d'ordre UBL

D.3.1 Exemple d'ordre d'assemblage

D.3.1.1 L'exemple d'ordre UBL est tiré de [1]. Précisément, l'exemple d'ordre d'assemblage a été choisi (voir xml/joinery/UBL-Order-1.0-Joinery-Example.xml) pour les raisons suivantes:

a) c'est un exemple réel développé indépendamment de la présente Recommandation | Norme internationale sans biais particulier à l'égard de Fast Infoset;

b) il est librement disponible;

c) il fait un large usage des espaces de nom XML et est donc un bon exemple pour présenter comment Fast Infoset prend en charge les espaces de nom XML.

D.3.1.2 L'exemple d'ordre d'assemblage a été modifié de la façon suivante:

a) les trois derniers éléments **OrderLine** ont été retirés;

NOTE 1 – Cela réduit le document XML 1.0 à une taille raisonnable pour sa présentation dans la présente Recommandation | Norme internationale.

b) tous les espaces blancs ont été retirés.

NOTE 2 – Ceci représente un cas d'utilisation plus réaliste pour les infosets XML qui peuvent être mis en série, transmis sur un réseau et analysés.

D.3.2 Document XML 1.0 en ordre d'assemblage

Le document XML 1.0 en ordre d'assemblage modifié comme indiqué au § D.3.1.2 a, mais avec les espaces blancs conservés pour améliorer la lisibilité, est présenté comme suit:

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:res="urn:oasis:names:tc:ubl:codelist:AcknowledgementResponseCode:1:0"
xmlns:cbc="urn:oasis:names:tc:ubl:CommonBasicComponents:1:0"
xmlns:cac="urn:oasis:names:tc:ubl:CommonAggregateComponents:1:0"
xmlns:cur="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oasis:names:tc:ubl:Order:1:0"
xsi:schemaLocation="urn:oasis:names:tc:ubl:Order:1:0 ../../xsd/maindoc/UBL-Order-1.0.xsd">
        <BuyersID>S03-034257</BuyersID>
        <cbc:IssueDate>2003-02-03</cbc:IssueDate>
        <cac:BuyerParty>
                <cac:Party>
                        <cac:PartyName>
                                <cbc:Name>Jerry Builder plc</cbc:Name>
                        </cac:PartyName>
                        <cac:Address>
                                <cbc:StreetName>Marsh Lane</cbc:StreetName>
                                <cbc:CityName>Nowhere</cbc:CityName>
                                <cbc:PostalZone>NR18 4XX</cbc:PostalZone>
                                <cbc:CountrySubentity>Norfolk</cbc:CountrySubentity>
                        </cac:Address>
                        <cac:Contact>
                                <cbc:Name>Eva Brick</cbc:Name>
                        </cac:Contact>
                </cac:Party>
        </cac:BuyerParty>
        <cac:SellerParty>
                <cac:Party>
```

```
                <cac:PartyName>
                        <cbc:Name>Specialist Windows plc</cbc:Name>
                </cac:PartyName>
                <cac:Address>
                        <cbc:BuildingName>Snowhill Works</cbc:BuildingName>
                        <cbc:CityName>Little Snoring</cbc:CityName>
                        <cbc:PostalZone>SM2 3NW</cbc:PostalZone>
                        <cbc:CountrySubentity>Whereshire</cbc:CountrySubentity>
                </cac:Address>
        </cac:Party>
</cac:SellerParty>
<cac:Delivery>
        <cbc:RequestedDeliveryDateTime>2003-02-24T00:00:00</cbc:RequestedDeliveryDateTime>
        <cac:DeliveryAddress>
                <cbc:StreetName>Riverside Rd.</cbc:StreetName>
                <cbc:BuildingName>Plot 17, Whitewater Estate</cbc:BuildingName>
                <cbc:CityName>Whetstone</cbc:CityName>
                <cbc:CountrySubentity>Middlesex</cbc:CountrySubentity>
        </cac:DeliveryAddress>
</cac:Delivery>
<cac:OrderLine>
        <cac:LineItem>
                <cac:BuyersID>A</cac:BuyersID>
                <cbc:Quantity quantityUnitCode="unit">2</cbc:Quantity>
                <cac:Item>
                        <cac:SellersItemIdentification>
                                <cac:ID>236WV</cac:ID>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>wood</cac:AttributeID>
                                        <cbc:Description>soft</cbc:Description>
                                </cac:PhysicalAttribute>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>finish</cac:AttributeID>
                                        <cbc:Description>primed</cbc:Description>
                                </cac:PhysicalAttribute>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>fittings</cac:AttributeID>
                                        <cbc:Description>satin</cbc:Description>
                                </cac:PhysicalAttribute>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>glazing</cac:AttributeID>
                                        <cbc:Description>single</cbc:Description>
                                </cac:PhysicalAttribute>
                        </cac:SellersItemIdentification>
                </cac:Item>
        </cac:LineItem>
</cac:OrderLine>
<cac:OrderLine>
        <cac:LineItem>
                <cac:BuyersID>B</cac:BuyersID>
                <cbc:Quantity quantityUnitCode="unit">3</cbc:Quantity>
                <cac:Item>
                        <cac:SellersItemIdentification>
                                <cac:ID>340TW</cac:ID>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>hand</cac:AttributeID>
                                        <cbc:Description>RH</cbc:Description>
                                </cac:PhysicalAttribute>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>wood</cac:AttributeID>
                                        <cbc:Description>hard</cbc:Description>
                                </cac:PhysicalAttribute>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>finish</cac:AttributeID>
                                        <cbc:Description>stain</cbc:Description>
                                </cac:PhysicalAttribute>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>fittings</cac:AttributeID>
                                        <cbc:Description>brass</cbc:Description>
                                </cac:PhysicalAttribute>
                                <cac:PhysicalAttribute>
                                        <cac:AttributeID>glazing</cac:AttributeID>
                                        <cbc:Description>double</cbc:Description>
                                </cac:PhysicalAttribute>
```

```
                                    </cac:SellersItemIdentification>
                                </cac:Item>
                            </cac:LineItem>
                        </cac:OrderLine>
                    </Order>
```

D.4 Document Fast Infoset en ordre UBL avec vocabulaire externe

Le vocabulaire externe du document Fast Infoset est présenté au § D.4.1. Les octets (comme caractères hexadécimaux) du document Fast Infoset sont présentés au § D.4.2. Des explications détaillées de certaines séquences d'octet du § D.4.2 sont présentées au § D.4.3. Le document Fast Infoset ne peut pas être considéré comme autodescriptif parce que des informations externes (le vocabulaire externe) sont nécessaires pour produire un infoset XML complet.

NOTE – Le document Fast Infoset peut toujours être traité par un analyseur grammatical Fast Infoset qui ne peut obtenir les tableaux de vocabulaire étant donné l'URI mais les indices de tableau de vocabulaire ne peuvent être déréférencés pour obtenir les informations nécessaires à la génération des propriétés des éléments d'information.

D.4.1 Le vocabulaire externe en ordre UBL

D.4.1.1 Le vocabulaire externe du document Fast Infoset est spécifié comme étant le vocabulaire final obtenu à partir de l'exemple d'infoset XML en ordre UBL (voir le § D.3.1.2) qui est ensuite modifié pour ne contenir:

a) aucun élément d'information **character**;

b) que des propriétés **[normalized value]** vides des éléments d'information **attribute**.

NOTE 1 – Ceci représente un scénario réaliste dans lequel on ne sait pas à l'avance ce que seront les propriétés du contenu défini par l'application (éléments d'information **character** et/ou propriétés **[normalized value]** des éléments d'information **attribute**) d'un infoset XML.

NOTE 2 – En pratique, on n'espère pas que le document à traiter servira à générer le vocabulaire externe. On prévoit que les outils utiliseront le schéma, et peut-être des instances d'infoset XML du schéma d'analyses de fréquence des chaînes et des noms qualifiés de sorte que de plus petites valeurs d'indice puissent être allouées aux informations survenant le plus fréquemment (par exemple, la fréquence des propriétés **[local name]** dans les infosets XML peut suivre une série de lois de puissance).

D.4.1.2 L'URI du vocabulaire externe est **urn:oasis:names:tc:ubl:Order:1.0:joinery:example**.

D.4.1.3 Le Tableau D.2 présente le vocabulaire de l'infoset XML en ordre UBL (les tableaux de vocabulaire). La colonne 1 liste les indices de tableau de vocabulaire des tableaux de vocabulaire (index), la colonne 2 liste les entrées de tableau de vocabulaire du tableau PREFIX (entrées de préfixes), la colonne 3 liste les entrées de tableau de vocabulaire du tableau NAMESPACE NAME (entrée de nom d'espace de nom), la colonne 4 liste les entrées de tableau de vocabulaire du tableau LOCAL NAME (entrée de nom local), la colonne 5 liste les entrées de tableau de vocabulaire du tableau ELEMENT NAME (entrée de nom d'élément), la colonne 6 liste les entrées de tableau de vocabulaire du tableau ATTRIBUTE NAME (entrée de nom d'attribut). Les valeurs d'indice pour les entrées de substitut de nom, des tableaux ELEMENT NAME et ATTRIBUTE NAME, sont présentées dans l'ordre spécifié pour le composant du type **`NameSurrogate`** (**`prefix-name-string-index`**, **`namespace-name-string-index`** et **`local-name-string-index`**). Un caractère "_" spécifie que la valeur est absente (ce qui n'arrive que pour les valeurs des composants **`prefix-name-string-index`** et **`namespace-name-string-index`**).

NOTE 1 – La première entrée (indice `1`) pour le préfixe et le nom d'espace de nom correspondant au préfixe XML, "xml", et le nom d'espace de nom XML, "http://www.w3.org/XML/1998/namespace", sont prédéfinis (voir les § 7.2.21 et 7.2.22).

NOTE 2 – Les entrées d'espaces de nom longs (URI) ont été tronquées.

NOTE 3 – Pour la première entrée de nom d'élément (indice `1`) il n'y a pas de référence à un préfixe (car la valeur est absente, représentée par "_"), il y a une référence à la septième entrée de nom d'espace de nom (indice `7`) pour la propriété **[namespace name]** ("urn:oasis:names:tc:ubl:Order:1:0"), et il y a une référence à la première entrée de nom local (indice `1`) pour la propriété **[local name]** ("Order").

Tableau D.2/X.891 – Vocabulaire de l'infoset XML en ordre UBL

Indice	Entrée de préfixe	Entrée de nom d'espace de nom	Entrée de nom local	Entrée de nom d'élément	Entrée de nom d'attribut
1	xml	http://www.w3.org/XML/1998/namespace	Order	_ 7 1	6 6 2
2	res	….AcknowledgementResponseCode:1:0	schemaLocation	_ 7 3	_ _ 23
3	cbc	….CommonBasicComponents:1:0	BuyersID	3 3 4	
4	cac	….CommonAggregateComponents:1:0	IssueDate	4 4 5	
5	cur	….CurrencyCode:1:0	BuyerParty	4 4 6	
6	xsi	….XMLSchema-instance	Party	4 4 7	
7		….Order:1:0	PartyName	3 3 8	
8			Name	4 4 9	
9			Address	3 3 10	
10			StreetName	3 3 11	
11			CityName	3 3 12	
12			PostalZone	3 3 13	
13			CountrySubentity	4 4 14	
14			Contact	4 4 15	
15			SellerParty	3 3 16	
16			BuildingName	4 4 17	
17			Delivery	3 3 18	
18			RequestedDeliveryDateTime	4 4 19	
19			DeliveryAddress	4 4 20	
20			OrderLine	4 4 21	
21			LineItem	4 4 3	
22			Quantity	3 3 22	
23			quantityUnitCode	4 4 24	
24			Item	4 4 25	
25			SellersItemIdentification	4 4 26	
26			ID	4 4 27	
27			PhysicalAttribute	4 4 28	
28			AttributeID	3 3 29	
29			Description		

D.4.2 Octets (comme caractères hexadécimaux) du document Fast Infoset

Le Tableau D.3 présente les octets du document Fast Infoset pour l'exemple d'ordre UBL présenté au § D.3.

NOTE – Les caractères hexadécimaux contenant des bits qui correspondent à l'identification et la terminaison des éléments d'information sont soulignés.

Tableau D.3/X.891 – Octets (comme caractères hexadécimaux) du document Fast Infoset

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000000	e0010000020100 02f75726e3a6f617369733a6e616d65733a74633a75626c3a4f
000020	726465723a313a303a6a6f696e6572793a6578616d706c6578cf8181cf8282cf
000040	8383cf8484cf8585cd86f00000083b75726e3a6f617369733a6e616d65733a74
000060	633a75626c3a4f726465723a313a30202e2e2f2e2e2f7873642f6d61696e646f
000080	632f55424c2d4f726465722d312e302e787364f00182075330332d3033343235
0000a0	37f00282073230303332d30322d3033f003040506820e4a65727279204275696c
0000c0	64657220706c63ff070882074d61727368204c616e65f00982044e6f77686572
0000e0	65f00a82054e52313820345858f00b82044e6f72666f6c6bff0c068206457661
000100	20427269636bffff0d040506821353706563696 16c6973742057696e646f7773
000120	20706c63ff070e820b536e6f7768696c6c20576f726b73f009820b4c6974746c
000140	6520536e6f72696e67f00a8204534d3220334e57f00b82075768657265736869
000160	7265ffff0f10821032303033 2d30322d32345430303a30303a3030f01108820a
000180	52697665727369646520526 42ef00e8217506c6f742031372c20576869746577
0001a0	617465722045737461746 5f00982065768657473746f6e65f00b82064d696464
0001c0	6c65736578fff01213149041f0550143756e6974f09032f01617189202323336
0001e0	5756f0191a9201776f6f64f01b9201736f6674ff191a820366696e697368f01b
000200	82037072696d6564ff191a820566697474696e6773f01b9202736174696eff19
000220	1a8204676c617a696e67f01b820373696e676c65ffffff1213149042f0550180
000240	f09033f016171892023334305457f0191a920168616e64f01b915248ff191aa3
000260	f01b920168617264ff191a820366696e697368f01b9202737461696eff191a82
000280	0566697474696e6773f01b9202627261737 3ff191a8204676c617a696e67f01b
0002a0	8203646f75626c65ffffffff
0002ac	

D.4.3 Explication du codage

D.4.3.1 Codage des éléments d'information document et Order element

Les explications suivantes précisent le codage initial du document Fast Infoset (y compris l'URI du vocabulaire externe) et l'élément d'information racine. Sont expliqués en particulier le codage d'un élément d'information **document**, d'une séquence d'éléments d'information **namespace**, d'un élément d'information **element**, et d'un élément d'information **attribute**. Le Tableau D.4 présente un fragment de document Fast Infoset pour le codage de l'élément d'information **document** et de l'élément d'information **Order element** du § D.3.2. Le Tableau D.5 détaille ce codage. Le fragment en XML 1.0 est présenté comme suit:

```
<Order xmlns:res="urn:oasis:names:tc:ubl:codelist:AcknowledgementResponseCode:1:0"
xmlns:cbc="urn:oasis:names:tc:ubl:CommonBasicComponents:1:0"
xmlns:cac="urn:oasis:names:tc:ubl:CommonAggregateComponents:1:0"
xmlns:cur="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oasis:names:tc:ubl:Order:1:0"
xsi:schemaLocation="urn:oasis:names:tc:ubl:Order:1:0 ../../xsd/maindoc/UBL-Order-1.0.xsd">
```

Tableau D.4/X.891 – Octets (comme caractères hexadécimaux) du fragment

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000000	e0010000020100 02f75726e3a6f617369733a6e616d65733a74633a75626c3a4f
000020	726465723a313a303a6a6f696e6572793a6578616d706c6578cf8181cf8282cf
000040	8383cf8484cf8585cd86f00000083b75726e3a6f617369733a6e616d65733a74
000060	633a75626c3a4f726465723a313a30202e2e2f2e2e2f7873642f6d61696e646f
000080	632f55424c2d4f726465722d312e302e787364f0

Tableau D.5/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
00 01	11100000 (e0) 00000000 (00)	Ces octets sont présents au début de chaque document Fast Infoset (voir § 12.6).	Elément d'information **document**
02 03	00000000 (00) 00000001 (01)	Ces octets sont le codage du numéro de version (voir § 12.9).	
04 05 06	00100000 (20) 00010000 (10) 00000000 (00)	Ces octets sont le codage de présence d'un vocabulaire initial et une référence à un vocabulaire externe du vocabulaire initial. L'octet de position 04_{16}, valeur 20_{16}, a un '0' (bourrage) au premier bit (voir § 12.8). Le troisième bit est '1' pour noter que le composant **initial-vocabulary** est présent, et que les six autres composants facultatifs sont absents (voir § C.2.3). L'octet de position 05_{16}, valeur 10_{16}, a trois '0' (bourrage) aux trois premiers bits (voir § C.2.5). Le quatrième bit est '1' pour noter que le **external-vocabulary** du **initial-vocabulary** est présent. Le quatre derniers bits sont '0' (bits cinq à huit) notant que quatre des douze autres composants facultatifs sont absents (voir § C.2.5.1). L'octet de position 06_{16}, valeur 00_{16}, a '0' pour tous les bits notant que les huit derniers des douze composants facultatifs sont absents (voir § C.2.5.1).	
07 08 37	00101111 (2f) 01110101 (75) 01100101 (65)	Ces octets sont le codage de l'URI du vocabulaire externe. L'octet de position 07_{16}, valeur $2f_{16}$, a un '0' (bourrage) au premier bit (voir § C.2.5.2). L'URI est codé comme caractères UTF-8 (voir § C.22). Le second bit est '0' notant que la longueur de l'URI est supérieure ou égale à 1_{10} octet et inférieure ou égale à 64_{10} octets, et que la longueur, moins la limite inférieure, est codée aux bits trois à huit comme un entier non signé (voir § C.22.3.1). L'entier non signé est 47_{10} et la longueur est 48_{10} (la limite inférieure est 1). Les 48_{10} octets des caractères codés en UTF-8 (de l'URI) sont codés à partir de l'octet en position 08_{16} jusqu'à l'octet en position 37_{16}.	
38	01111000 (78)	Cet octet est le codage initial d'un fils de l'élément d'information **document**. L'octet en position 38_{16}, valeur 78_{16}, a '0' (identification) au premier bit notant qu'il y a un fils de l'élément d'information **document**, et le fils est un élément d'information **element** (voir § C.2.11.2). Le second bit est '1' notant que l'élément d'information **element** a des attributs (voir § C.3.3). Les bits trois à six sont '1110' suivis de '00' (bourrage) aux septième et huitième bits, notant que des éléments d'information **attribute** d'espace de nom sont présents (voir § C.3.4.1).	Elément d'information **element** avec propriété **[namespace attribute]**
39 3a 3b	11001111 (cf) 10000001 (81) 10000001 (81)	Ces octets sont le codage de l'élément d'information **attribute** d'espace de nom avec les propriétés **[prefix]** et **[normalized value]** indexées. L'octet de position 39_{16}, valeur cf_{16}, a '110011' (identification) aux six premiers bits (du premier au cinquième bit) notant qu'un élément d'information **attribute** d'espace de nom est présent (voir § C.3.4.2). Le septième bit est '1' notant que la propriété **[prefix]** est présente. Le huitième bit est '1' notant que la propriété **[normalized value]** est présente. L'octet de position $3a_{16}$, valeur 81_{16}, a '1' au premier bit notant qu'un indice est codé, et que l'indice dans le tableau PREFIX identifiera la propriété **[prefix]** (voir § C.13.4). Le second bit est '0' notant que l'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 64_{10}, et que l'indice est codé des bits trois à huit comme entier non signé (voir § C.25.2). L'entier non signé est 1_{10} et l'indice est 2_{10} (la limite inférieure est 1_{10}), ce qui donne la propriété **[prefix]** "res" lors du déréférencement du tableau PREFIX. L'octet de position $3b_{16}$, valeur 81_{16}, a '1' au premier bit notant qu'un indice est codé, et que l'indice dans le tableau NAMESPACE NAME identifiera la propriété **[normalized value]** (voir § C.13.4). Le second bit est '0' notant que l'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 64_{10}, et l'indice est codé des bits trois à huit comme entier non signé (voir § C.25.2). L'entier non signé est 1_{10} et l'indice est 2_{10} (la limite inférieure est 1_{10}), ce qui donne la propriété **[normalized value]** ".ResponseCode:1.0" lors du déréférencement du tableau NAMESPACE NAME.	**xmlns:res= "….ResponseCode:1.0"**

Tableau D.5/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
3c 3d 3e	11001111 (cf) 10000010 (82) 10000010 (82)	Ces octets sont le codage de l'élément d'information **attribute** d'espace de nom avec les propriétés **[prefix]** et **[normalized value]**. L'indice de la propriété **[prefix]** est 3_{10}, ce qui donne une valeur de "cbc" lors du déréférencement du tableau PREFIX. L'indice de la propriété **[normalized value]** est 3_{10}, ce qui donne une valeur de "….sicComponents:1.0" lors du déréférencement du tableau NAMESPACE NAME.	**xmlns:cbc= "….sicComponents:1:0"**
3f 40 41	11001111 (cf) 10000011 (83) 10000011 (83)	Ces octets sont le codage de l'élément d'information **attribute** d'espace de nom avec les propriétés **[prefix]** et **[normalized value]**.	**xmlns:cac= "….ateComponents:1:0"**
42 43 44	11001111 (cf) 10000100 (84) 10000100 (84)	Ces octets sont le codage de l'élément d'information **attribute** d'espace de nom avec les propriétés **[prefix]** et **[normalized value]**.	**xmlns:cur= "….CurrencyCode:1:0"**
45 46 47	11001111 (cf) 10000101 (85) 10000101 (85)	Ces octets sont le codage de l'élément d'information **attribute** d'espace de nom avec les propriétés **[prefix]** et **[normalized value]**.	**xmlns:xsi= "….Schema-instance"**
48 49	11001101 (cd) 10000110 (86)	Ces octets sont le codage de l'élément d'information **attribute** d'espace de nom avec une propriété **[normalized value]** indexée. L'octet de position 48_{16}, valeur cd_{16}, a un septième bit de '0' notant l'absence de la propriété **[prefix]**, et un huitième bit de '1' notant la présence de la propriété **[normalized value]**.	**xmlns="….Order:1:0"**
4a	11110000 (f0)	Cet octet est le codage de la terminaison pour la séquence des éléments d'information **attribute** d'espace de nom. L'octet de position $4a_{16}$, valeur $f0_{16}$, a '1111' (terminaison) aux quatre premiers bits (du premier au quatrième bit) et c'est la terminaison de la séquence. Quatre '0' (bourrage) sur les six sont présents du cinquième au huitième bit (voir § C.3.4.3).	
4b	00000000 (00)	Cet octet est le codage d'un nom qualifié indexé de l'élément d'information **element**. L'octet de position $4b_{16}$, valeur, 00_{16}, a les deux derniers des six '0' (bourrage) sur les premier et second bits (voir § C.3.4.3). Le troisième bit est '0' notant que le nom qualifié n'est pas un nom qualifié litéral (voir § C.18.3) et qu'il est indexé. L'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 32_{10}, et l'indice est codé des bits quatre à huit comme un entier non signé (§ C.27.2). L'entier non signé est 0_{10} et l'indice est 1_{10} (la limite inférieure est 1_{10}), ce qui donne un nom qualifié avec une propriété **[namespace name]** de "….Order:1.0" et une propriété **[local name]** de "Order" (il n'y a pas de propriété **[prefix]** pour ce nom qualifié) lors du déréférencement du tableau ELEMENT NAME.	**<Order ….**

Tableau D.5/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
4c 4d 4e 4f 92	00000000 (00) 00001000 (08) 01111011 (3b) 01110101 (75) 01101000 (64)	Ces octets sont le codage d'un élément d'information **attribute** avec un nom qualifié indexé et une propriété **[normalized value]**. La présence d'éléments d'information **attribute** était notée dans l'octet de position 38_{16} (le second bit est '1'). L'octet de position $4c_{16}$, valeur 00_{16}, a un premier bit de '0' (identification) notant la présence d'un élément d'information **attribute** (voir § C.3.6.1). Le second bit est '0' notant que le nom qualifié n'est pas un nom qualifié littéral (voir § C.17.3) et est indexé. L'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 64_{10}, et l'indice est codé des bits trois à huit comme un entier arithmétique (voir § C.25.2). L'entier arithmétique est 0_{10} et l'indice est 1_{10} (la limite inférieure est 1_{10}), ce qui donne un nom qualifié avec une propriété **[prefix]** de "xsi", une propriété **[namespace name]** de "….Schema-instance" et une propriété **[local name]** de "schemaLocation" lors du déréférencement du tableau ATTRIBUTE NAME. L'octet de position $4d_{16}$, valeur 08_{16}, est le codage initial d'une chaîne ou indice non identifiant (voir § C.14) pour la propriété **[normalized value]**. Le premier bit est '0' notant la présence d'une chaîne de caractères littérale (voir § C.14.3). Le second bit est '0' notant que la chaîne de caractères littérale ne devrait pas être ajoutée au tableau ATTRIBUTE VALUE. Les bits trois et quatre, tous deux à '0', notent que le format de codage de la chaîne est en UTF-8 (voir § C.19.3.1). Les bits cinq et six sont à '1' et '0' respectivement, ce qui note que la longueur des octets des caractères codés en UTF-8 (la propriété **[normalized value]**) est supérieure ou égale à 9_{10} octets et inférieure ou égale à 264_{10} octets, et que la longueur, moins la limite inférieure, est codée en huit bits sur le prochain octet comme un entier non signé (voir § C.23.3.2). Les bits sept et huit sont à '0' (bourrage) (voir § C.23.3.2). L'octet de position $4e_{16}$, valeur $3b_{16}$, est le codage de l'entier non signé. La longueur des octets des caractères codés en UTF-8 est 68_{10} (la limite inférieure est 9_{10}). Les 68_{10} octets des caractères codés en UTF-8 (de la propriété **[normalized value]**) sont codés à partir de l'octet de position $4f_{16}$ jusqu'à l'octet de position 92_{16}.	**xsi:schemaLocation="…."**
93	11110000 (f0)	Cet octet est le codage de la terminaison pour la séquence d'éléments d'information **attribute**. L'octet de position 93_{16}, valeur $f0_{16}$, a '1111' sur les quatre premiers bits (du premier au quatrième bit) et c'est la terminaison pour la séquence. Quatre '0' (bourrage) sont présents (du bit cinq au bit huit) car l'élément d'information **Order element** a des fils (voir § D.3.2).	

D.4.3.2 Codage de l'élément d'information Address element de l'élément d'information BuyerParty element

Les explications suivantes détaillent le codage de l'élément d'information **Address element** de l'élément d'information **BuyerParty element** du document Fast Infoset. En particulier, le codage de l'élément d'information **element** et de l'élément d'information **character** est expliqués. Le Tableau D.6 présente un fragment de document Fast Infoset pour le codage de l'élément d'information **Address element** de l'élément d'information **BuyerParty element** du § D.3.2. Le Tableau D.7 donne le détail de ce codage. Le fragment en XML 1.0 se présente comme suit:

```
<cac:Address>
        <cbc:StreetName>Marsh Lane</cbc:StreetName>
        <cbc:CityName>Nowhere</cbc:CityName>
        <cbc:PostalZone>NR18 4XX</cbc:PostalZone>
        <cbc:CountrySubentity>Norfolk</cbc:CountrySubentity>
</cac:Address>
```

Tableau D.6/X.891 – Octets (comme caractères hexadécimaux) du fragment

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
0000c0	070882074d61727368204c616e65f00982044e6f77686572
0000e0	65f00a82054e52313820345858f00b82044e6f72666f6c6bff

Tableau D.7/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
c8	00000111 (07)	Cet octet est le codage de l'élément d'information **Address element**. L'octet de position $c8_{16}$, valeur 07_{16}, a '0' (identification) au premier bit notant qu'il y a un fils d'un élément d'information **element** (fils de l'élément d'information **Party element**), et le fils est un élément d'information **element** (voir § C.3.7.2). Le second bit est '0' notant que l'élément d'information **element** n'a pas d'attribut (voir § C.3.3). Le troisième bit est '0' notant que le nom qualifié n'est pas un nom qualifié littéral (voir § C.18.3) et est indexé. L'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 32_{10}, et l'indice est codé des bits quatre à huit comme un entier non signé (voir § C.27.2). L'entier non signé est 7_{10} et l'indice est 8_{10} (la limite inférieure est 1_{10}), ce qui donne un nom qualifié avec un **[prefix]** de "cac", un propriété **[namespace name]** de "….gateComponents:1.0" et une propriété **[local name]** de "Address" lors du déréférencement du tableau ELEMENT NAME.	**<cac:Address>**
c9	00001000 (08)	Cet octet est le codage de l'élément d'information **StreetName element**. L'élément d'information **element** a un indice de 9_{10}, qui donne un nom qualifié avec un **[prefix]** de "cbc", une propriété **[namespace name]** de "….BasicComponents:1:0" et une propriété **[local name]** de "StreetName" lors du déréférencement du tableau ELEMENT NAME.	**<cbc:StreetName>**
ca cb cc d5	10000010 (82) 00000111 (07) 01001101 (4d) 01100101 (65)	Ces octets sont le codage de l'élément d'information **character** de l'élément d'information **StreetName element**. L'octet de position ca_{16}, valeur 82_{16}, a '10' (identification) aux deux premiers bits (les bits un et deux) notant qu'il y a un fils de l'élément d'information **element** (fils de l'élément d'information **StreetName element**), et le fils est un tronçon des éléments d'information **character** (voir § C.3.7.5). Le troisième bit est '0' notant qu'une chaîne de caractères littérale est présente (voir § C.15.3). Le quatrième bit est '0' notant que la chaîne de caractères littérale ne devrait pas être ajoutée au tableau CONTENT CHARACTER CHUNK. Les bits cinq et six, tous deux à '0', notent que le format de codage du tronçon est en UTF-8 (voir § C.20.3.1). Les bits sept et huit sont respectivement '1' et '0', notant que la longueur des octets des caractères codés en UTF-8 (le tronçon de l'élément d'information **character**) est supérieur ou égal à 3_{10} octets et inférieur ou égal à 258_{10} octets, et que la longueur, moins la limite inférieure, est codée sur huit bits du prochain octet comme un entier non signé (voir § C.24.3.2). L'octet de position cb_{16}, valeur 07_{16}, est l'entier non signé. La longueur des octets des caractères codés en UTF-8 est 10_{10} (la limite inférieure est 3_{10}). Les 10_{10} octets des caractères codés en UTF-8 sont codés depuis l'octet de position cc_{10} jusqu'à l'octet de position $d5_{10}$.	Eléments d'information **character** "Marsh Lane"
d6	11110000 (f0)	Cet octet est la terminaison pour l'élément d'information **StreetName element**. L'octet de position $d6_{16}$, valeur $f0_{16}$, a '1111' (terminaison) aux quatre premiers bits (du bit un au bit quatre) et est la terminaison pour l'élément d'information **StreetName element** (voir § C.3.8). Les bits cinq à huit sont '0' (bourrage) si le futur fils (homologue) survient (élément d'information **CityName element**) (voir § C.3.7.1).	**</cbc:StreetName>**
d7	00001001 (09)	Cet octet est le codage de l'élément d'information **CityName element**. L'élément d'information **element** a un indice de 10_{10}, qui donne un nom qualifié avec un **[prefix]** de "cbc", une propriété **[namespace name]** de "….BasicComponents:1:0" et une propriété **[local name]** de "CityName" lors du déréférencement du tableau ELEMENT NAME.	**<cbc:CityName>**
d8 d9 da	10000010 (82) 00000100 (04) 01001110 (4e)	Ces octets sont le codage de l'élément d'information **character** de l'élément d'information **CityName element**. Les 7_{10} octets des caractères codés en UTF-8 sont codés à partir de l'octet de position da_{16} jusqu'à l'octet de position $e0_{16}$.	Eléments d'information **character** "Nowhere"

Tableau D.7/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
e0	01100101 (65)		
e1	1111<u>0000</u> (f0)	Cet octet est la terminaison pour l'élément d'information **CityName element**.	**</cbc:CityName>**
e2	00001010 (0a)	Cet octet est le codage de l'élément d'information **PostalZone element**. L'élément d'information **element** a un indice de 11_{10}, qui donne un nom qualifié avec un **[prefix]** de "cbc", une propriété **[namespace name]** de "….BasicComponents:1:0" et une propriété **[local name]** de "PostalZone" lors du déréférencement du tableau ELEMENT NAME.	**<cbc:PostalZone>**
e3 e4 e5 .... ec	10000010 (82) 00000101 (05) 01001110 (4e) 01011000 (58)	Ces octets sont le codage de l'élément d'information **character** de l'élément d'information **PostalZone element**. Les 8_{10} octets des caractères codés en UTF-8 sont codés depuis l'octet de position $e5_{16}$ jusqu'à l'octet de position ec_{16}.	Eléments d'information **character** "NR18 4XX"
ed	1111<u>0000</u> (f0)	Cet octet est la terminaison pour l'élément d'information **PostalZone element**.	**</cbc:PostalZone>**
ee	00001011 (0b)	Cet octet est le codage de l'élément d'information **CountrySubentity element**. L'élément d'information **element** a un indice de 12_{10}, qui donne un nom qualifié avec un **[prefix]** de "cbc", une propriété **[namespace name]** de "….BasicComponents:1:0" et une propriété **[local name]** de "CountrySubentity" lors du déréférencement du tableau ELEMENT NAME.	**<cbc:CountrySubentity>**
ef f0 f1 .... f7	10000010 (82) 00000100 (04) 01001110 (4e) 01101011 (6b)	Ces octets sont le codage des éléments d'information **character** de l'élément d'information **CountrySubentity element**. Les 7_{10} octets des caractères codés en UTF-8 sont codés depuis l'octet de position $f1_{16}$ jusqu'à l'octet de position $f7_{16}$.	Eléments d'information **character** "Norfolk"
f8	11111111 (ff)	Cet octet est la terminaison pour l'élément d'information **CountrySubentity element** et l'élément d'information **Address element**. L'octet de position $f8_{16}$, valeur ff_{16}, a '1111' (terminaison) aux quatre premiers bits (les bits un à quatre) et c'est la terminaison pour l'élément d'information **CountrySubentity element** (voir § C.3.8). Les quatre derniers bits (les bits cinq à huit) sont '1111' et c'est la terminaison pour l'élément d'information **Address element** (voir § C.3.8).	**</cbc:CountrySubentity>** **</cac:Address>**

D.5 Document Fast Infoset en ordre UBL sans vocabulaire initial

Ces octets (comme caractères hexadécimaux) du document Fast Infoset sont présentés au § D.5.1. Des explications détaillées de certaines séquences d'octet du § D.5.1 sont présentées au § D.5.2. Le vocabulaire final de ce document Fast Infoset et celui de l'ancien document Fast Infoset seront identiques dans la mesure où les indices de tableau de vocabulaire des tableaux dans le vocabulaire externe sont générés dans le même ordre. Comme les chaînes sont incorporées dans le document Fast Infoset, il en résultera une plus grande taille. Le coût de l'inclusion des chaînes est de 635_{10} octets (la taille de ce document Fast Infoset moins la taille de l'ancien document Fast Infoset), ce qui fait approximativement la moitié de la taille du document (pour des documents plus grands, cette différence devrait être inférieure car le vocabulaire tendra vers un coût fixe). A la différence de l'ancien document Fast Infoset, ce document peut être considéré comme autodescriptif parce que l'infoset XML peut être produit sans aucune information externe (pas de vocabulaire externe).

D.5.1 Octets (comme caractères hexadécimaux) du document Fast Infoset

Le Tableau D.8 présente les octets du document Fast Infoset pour l'exemple d'ordre UBL présenté au § D.3.

NOTE – Les caractères hexadécimaux qui contiennent des bits correspondant à l'identification et la terminaison des éléments d'information sont soulignés.

Tableau D.8/X.891 – Octets (comme caractères hexadécimaux) du document Fast Infoset

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000000	e00100000078cf027265733e75726e3a6f617369733a6e616d65733a74633a75
000020	626c3a636f64656c6973743a41636b6e6f776c656467656d656e74526573706f
000040	6e7365436f64653a313a30cf026362632f75726e3a6f617369733a6e616d6573
000060	3a74633a75626c3a436f6d6d6f6e4261736963436f6d706f6e656e74733a313a
000080	30cf026361633375726e3a6f617369733a6e616d65733a74633a75626c3a436f
0000a0	6d6d6f6e41676772656761746543 6f6d706f6e656e74733a313a30cf02637572
0000c0	2f75726e3a6f617369733a6e616d65733a74633a75626c3a636f64656c697374
0000e0	3a43757272656e6379436f64653a313a30cf027873692868747 4703a2f2f7777
000100	772e77332e6f72672f323030312f584d4c536368656d612d696e7374616e6365
000120	cd1f75726e3a6f617369733a6e616d65733a74633a75626c3a4f726465723a31
000140	3a30f03d86044f726465727b85850d736368656d614c6f636174696f6e083b75
000160	726e3a6f617369733a6e616d65733a74633a75626c3a4f726465723a313a3020
000180	2e2e2f2e2e2f7873642f6d61696e646f632f55424c2d4f726465722d312e302e
0001a0	787364f03d860742757965727349448207533033 2d303334323537f03f828208
0001c0	4973737565446174658207323030332d30322d3033f03f838309427579657250
0001e0	617274793f838304506172747 4793f8383085061727479 4e616d653f8282034e61
000200	6d65820e4a65727279204275696c64657220706c63ff3f838306416464726573
000220	733f828209537472656574e616d6582074d61727368204c616e65f03f828207
000240	436974794e616d6582044e6f7768657265f03f828209506f7374616c5a6f6e65
000260	82054e52313820345858f03f82820f436f756e747279537562656e7469747982
000280	044e6f72666f6c6bff3f838306436f6e7461637406820645766120427269636b
0002a0	ffff3f83830a53656c6c6572506172747904050682135370656369616c697374
0002c0	2057696e646f777320706c63ff073f82820b4275696c64696e674e616d65820b
0002e0	536e6f7768696c6c20576f726b73f009820b4c6974746c6520536e6f72696e67
000300	f00a8204534d3220334e57f00b820757686572657368697265ffff3f83830744
000320	656c69766572793f828218526571756573746564446 56c697665727944617465
000340	54696d6582103230303332d30322d32345430303a30303a3030f03f83830e4465
000360	6c6976657279416464726573730882 0a52697665727369646520526 42ef00e82
000380	17506c6f742031372c20576869746577617465722045737461746 5f009820657
0003a0	68657473746f6e65f00b82064d6964646c65736578fff03f8383084f72646572
0003c0	4c696e653f8383074c696e654974656d3f8383829041f07f8282075175616e74
0003e0	697479780f7175616e74697479556e6974436f646543756e6974f09032f03f83
000400	83034974656d3f83831853656c6c6572734974656d4964656e74696669636174
000420	696f6e3f838301494492023233365756f03f838310506879736963616c417474
000440	7269627574653f83830a41747472696275746549449201776f6f64f03f82820a
000460	4465736372697074696f6e9201736f6674ff191a820366696e697368f01b8203
000480	7072696d6564ff191a820566697474696e6773f01b9202736174696eff191a82
0004a0	04676c617a696e67f01b820373696e676c65ffffff1213149042f0550180f090
0004c0	33f016171892023334305457f0191a920168616e64f01b915248ff191aa3f01b
0004e0	920168617264ff191a820366696e697368f01b9202737461696eff191a820566
000500	697474696e6773f01b92026272617373ff191a8204676c617a696e67f01b8203
000520	646f75626c65ffffffff
00052a	

D.5.2 Explication du codage

D.5.2.1 Codage de l'élément d'information document et de l'élément d'information Order element

Les explications suivantes détaillent le codage initial du document Fast Infoset et de l'élément d'information racine. En particulier sont expliqués le codage d'un élément d'information **document**, d'une séquence d'éléments d'information **namespace**, d'un élément d'information **element**, et d'un élément d'information **attribute**. Le Tableau D.9 présente le fragment du document Fast Infoset pour le codage de l'élément d'information **document** et de l'élément d'information

Order element du § D.3.2. Le Tableau D.10 donne le détail de ce codage. Le fragment en XML 1.0 est présenté comme suit:

```
<Order xmlns:res="urn:oasis:names:tc:ubl:codelist:AcknowledgementResponseCode:1:0"
xmlns:cbc="urn:oasis:names:tc:ubl:CommonBasicComponents:1:0"
xmlns:cac="urn:oasis:names:tc:ubl:CommonAggregateComponents:1:0"
xmlns:cur="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oasis:names:tc:ubl:Order:1:0"
xsi:schemaLocation="urn:oasis:names:tc:ubl:Order:1:0 ../../xsd/maindoc/UBL-Order-1.0.xsd">
```

Tableau D.9/X.891 – Octets comme caractères hexadécimaux) du fragment

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000000	e00100000078cf027265733e75726e3a6f617369733a6e616d65733a74633a75
000020	626c3a636f64656c6973743a41636b6e6f776c656467656d656e74526573706f
000040	6e7365436f64653a313a30cf026362632f75726e3a6f617369733a6e616d6573
000060	3a74633a75626c3a436f6d6d6f6e4261736963436f6d706f6e656e74733a313a
000080	30cf026361633375726e3a6f617369733a6e616d65733a74633a75626c3a436f
0000a0	6d6d6f6e41676772656761746543 6f6d706f6e656e74733a313a30cf02637572
0000c0	2f75726e3a6f617369733a6e616d65733a74633a75626c3a636f64656c697374
0000e0	3a43757272656e6379436f64653a313a30cf027873692868747 4703a2f2f7777
000100	772e77332e6f72672f323030312f584d4c536368656d612d696e7374616e6365
000120	cd1f75726e3a6f617369733a6e616d65733a74633a75626c3a4f726465723a31
000140	3a30f03d86044f726465727b85850d736368656d614c6f636174696f6e083b75
000160	726e3a6f617369733a6e616d65733a74633a75626c3a4f726465723a313a3020
000180	2e2e2f2e2e2f7873642f6d61696e646f632f55424c2d4f726465722d312e302e
0001a0	787364f0

Tableau D.10/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
00 01	11100000 (e0) 00000000 (00)	Ces octets sont présents au début de chaque document Fast Infoset (voir § 12.6).	Elément d'information **document**
02 03	00000000 (00) 00000001 (01)	Ces octets sont le codage du numéro de version (voir § 12.9).	
04	00000000 (00)	Ces octets sont le codage de la présence d'un vocabulaire initial et des autres composants du type **Document**. L'octet de position 04_{16}, valeur 00_{16}, a un '0' (bourrage) au premier bit (voir § 12.8). Les bits du second au huitième sont '0000000' notant que tous les composants facultatifs du type **Document** sont absents (y compris le composant **initial-vocabulary** dont l'absence est notée au troisième bit, voir § C.2.3).	
05	01111000 (78)	Cet octet est le codage initial d'un fils de l'élément d'information **document**. L'octet de position 05_{16}, valeur 78_{16}, a un '0' (identification) au premier bit notant qu'il y a un fils de l'élément d'information **document**, et que le fils est un élément d'information **element** (voir § C.2.11.2). Le second bit est '1' notant que l'élément d'information **element** a des attributs (voir § C.3.3). Les bits trois à six sont '1110' suivis par '00' (bourrage) sur les bits sept et huit, notant que des éléments d'information namespace **attribute** sont présents (voir § C.3.4.1).	Elément d'information **element** avec propriété **[namespace attribute]**.

Tableau D.10/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
06 07 08 .... 0a 0b 0c 4a	11001111 (cf) 00000010 (02) 01110010 (72) .... 01110010 (73) 00111110 (3e) 01110101 (75) 01110000 (30)	Ces octets sont le codage de l'élément d'information namespace **attribute** avec les propriétés littérales **[prefix]** et **[normalized value]**. L'octet de position 06_{16}, valeur cf_{16}, a '110011' (identification) du premier au sixième bit, notant qu'un élément d'information namespace **attribute** est présent (voir § C.3.4.2). Le septième bit est '1' notant que la propriété **[prefix]** est présente. Le huitième bit est '1' notant que la propriété **[normalized value]** est présente. L'octet de position 07_{16}, valeur 02_{16}, a '0' au premier bit notant qu'une chaîne de caractères littérale est codée pour la propriété **[prefix]** (voir § C.13.3). Le second bit est '0' notant que la longueur des caractères codés en UTF-8 est supérieure ou égale à 1_{10} et inférieure ou égale à 64_{10}, et la longueur est codée des bits trois à huit comme un entier non signé (voir § C.22.3.1). L'entier non signé est 2_{10} et la longueur est 3_{10} (la limite inférieure est 1_{10}). Les 3_{10} octets des caractères codés en UTF-8 (de la propriété **[prefix]**) sont codés depuis l'octet de position 08_{16} jusqu'à l'octet de position $0a_{10}$. La chaîne "res" sera ajoutée au tableau PREFIX (avec un indice de 2_{10}). L'octet de position $0b_{16}$, valeur $3e_{16}$, a '0' au premier bit notant qu'une chaîne de caractères littérale est codée pour la propriété **[normalized value]** (voir § C.13.3). Le second bit est '0' notant que la longueur des caractères codés en UTF-8 est supérieure ou égale à 1_{10} et inférieure ou égale à 64_{10}, et la longueur est codée des bits trois à huit comme un entier non signé (voir § C.22.3.1). L'entier non signé est 62_{10} et la longueur est 63_{10} (la limite inférieure est 1_{10}). Les 63_{10} octets des caractères codés en UTF-8 (de la propriété **[normalized value]**) sont codés depuis l'octet de position $0c_{16}$ jusqu'à l'octet de position $4a_{16}$. La chaîne "….ResponseCode:1:0" sera ajoutée au tableau NAMESPACE NAME (avec un indice de 2_{10}).	**xmlns:res=** **"….ResponseCode:1:0"**
4b 4c 4d 4f 50 51 80	11001111 (cf) 00000010 (02) 01100011 (63) .... 01100011 (63) 00101111 (2f) 01110101 (75) 01110000 (30)	Ces octets sont le codage de l'élément d'information namespace **attribute** avec les propriétés littérales **[prefix]** et **[normalized value]**. Les 3_{10} octets des caractères codés en UTF-8 (de la propriété **[prefix]**) sont codés depuis l'octet de position $4c_{16}$ à l'octet de position $4f_{16}$. La chaîne "cbc" sera ajoutée au tableau PREFIX (avec un indice 3_{10}). Les 48_{10} octets des caractères codés en UTF-8 (de la propriété **[normalized value]**) sont codés depuis l'octet de position 51_{16} jusqu'à l'octet de position 80_{16}. La chaîne "….sicComponents:1:0" sera ajoutée au tableau NAMESPACE NAME (avec un indice de 3_{10}).	**xmlns:cbc=** **"….sicComponents:1:0"**
81 82 83 .... 85 86 87 .... ba	11001111 (cf) 00000010 (02) 01100011 (63) .... 01100011 (63) 00110011 (33) 01110101 (75) 01110000 (30)	Ces octets sont le codage de l'élément d'information namespace **attribute** avec les propriétés littérales **[prefix]** et **[normalized value]**. Les 3_{10} octets des caractères codés en UTF-8 (de la propriété **[prefix]**) sont codés depuis l'octet de position 83_{16} jusqu'à l'octet de position 85_{16}. La chaîne "cac" sera ajoutée au tableau PREFIX (avec un indice de 4_{10}). Les 52_{10} octets des caractères codés en UTF-8 (de la propriété **[normalized value]**) sont codés depuis l'octet de position 87_{16} jusqu'à l'octet de position ba_{16}. La chaîne "….ateComponents:1:0" sera ajoutée au tableau NAMESPACE NAME (avec un indice de 4_{10}).	**xmlns:cac=** **"….ateComponents:1:0"**

Tableau D.10/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
bb bc bd .... bf c0 c1 .... f0	11001111 (cf) 00000010 (02) 01100011 (63) .... 01100011 (63) 00101111 (2f) 01110101 (75) 01110000 (30)	Ces octets sont le codage de l'élément d'information namespace **attribute** avec les propriétés littérales **[prefix]** et **[normalized value]**. Les 3_{10} octets des caractères codés en UTF-8 (de la propriété **[prefix]**) sont codés depuis l'octet de position bd_{16} jusqu'à l'octet de position bf_{16}. La chaîne "cur" sera ajoutée au tableau PREFIX (avec un indice de 5_{10}). Les 48_{10} octets des caractères codés en UTF-8 (de la propriété **[normalized value]**) sont codés depuis l'octet de position $c1_{16}$ jusqu'à l'octet de position $f0_{16}$. La chaîne "….CurrencyCode:1:0" sera ajoutée au tableau NAMESPACE NAME (avec un indice de 5_{10}).	**xmlns:cur=** **"….CurrencyCode:1:0"**
f1 f2 f3 .... f5 f6 f7 .... 11f	11001111 (cf) 00000010 (02) 01111000 (78) .... 01101001 (69) 00101000 (28) 01101000 (68) 01110111 (77)	Ces octets sont le codage de l'élément d'information namespace **attribute** avec les propriétés littérales **[prefix]** et **[normalized value]** Les 3_{10} octets des caractères codés en UTF-8 (de la propriété **[prefix]**) sont codés depuis l'octet de position $f3_{16}$ jusqu'à l'octet de position $f5_{16}$. La chaîne "xsi" sera ajoutée au tableau PREFIX (avec un indice de 6_{10}). Les 41_{10} octets des caractères codés en UTF-8 (de la propriété **[normalized value]**) sont codés depuis l'octet de position $f7_{16}$ jusqu'à l'octet de position $11f_{16}$. La chaîne "….Schema-instance" sera ajoutée au tableau NAMESPACE NAME (avec un indice de 6_{10}).	**xmlns:xsi=** **"….Schema-instance"**
120 121 122 141	11001101 (cd) 00011111 (1f) 01110101 (75) 00110000 (30)	Ces octets sont le codage de l'élément d'information namespace **attribute** avec une propriété indexée **[normalized value]**. Les 32_{10} octets des caractères codés en UTF-8 (de la propriété **[normalized value]**) sont codés depuis l'octet de position 122_{16} à l'octet de position 141_{16}. La chaîne "….Order:1:0" sera ajoutée au tableau NAMESPACE NAME (avec un indice de 7_{10}).	**xmlns="….Order:1:0"**
142	1111<u>0000</u> (f0)	Cet octet est le codage de la terminaison pour la séquence d'éléments d'information **attribute** d'espace de nom. L'octet de position 142_{16}, valeur $f0_{16}$, a '1111' (terminaison) aux quatre premiers bits (les bits un à quatre) et c'est la terminaison de la séquence. Quatre des six '0' (bourrage) sont présents du cinquième au huitième bit (voir § C.3.4.3).	
143 144 145 146 .... 14a	<u>00</u>111101 (3d) 10000110 (86) 00000100 (04) 01001111 (4f) 01110010 (72)	Ces octets sont le codage d'un nom qualifié littéral de l'élément d'information **element**. L'octet de position 143_{16}, valeur $3d_{16}$, a les deux derniers des six '0' (bourrage) aux premier et second bits (voir § C.3.4.3). Les bits trois et quatre sont '1111', notant que le nom qualifié est un nom qualifié littéral (voir § C.18.3). Le septième bit est '0' notant que le nom qualifié n'a pas de propriété **[prefix]**. Le huitième bit est '1' notant que le nom qualifié a une propriété **[namespace name]**. L'octet de position 144_{16}, valeur 86_{16}, a '1' au premier bit notant que la propriété **[namespace name]** n'est pas une chaîne littérale et qu'elle est indexée (voir § C.13.4). Le second bit est '0' notant que l'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 64_{10}, et l'indice est codé des bits trois à huit comme un entier non signé (voir § C.25.2). L'entier non signé est 6_{10} et l'indice est 7_{10} (la limite inférieure est 1_{10}), ce qui donne une propriété **[namespace name]** de "….Order:1:0" lors du déréférencement du tableau NAMESPACE NAME. L'octet de position 145_{16}, valeur 04_{16}, a '0' au premier bit notant que cette chaîne de caractères littérale est codée pour la propriété **[local name]** (voir § C.13.3). Le second bit est '0' notant que la longueur des caractères codés en UTF-8 est supérieure ou égale à 1_{10} et inférieure ou égale à 64_{10}, et la longueur est codée des bits trois à huit comme un entier non signé (voir § C.22.3.1). L'entier non signé est 4_{10} et la longueur est 5_{10} (la limite inférieure est 1_{10}).	**<Order ….**

Tableau D.10/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
		Les 5_{10} octets des caractères codés en UTF-8 (de la propriété **[local name]**) sont codés depuis l'octet de position 146_{16} à l'octet de position $14a_{16}$. La chaîne "Order" sera ajoutée au tableau LOCAL NAME (avec un indice de 1_{10}). Le nom qualifié sans propriété **[prefix]**, une propriété **[namespace name]** de "….Order:1:0" (indice 7_{10}), et une propriété **[local name]** de "Order" (indice 1_{10}) sera ajouté au tableau ELEMENT NAME (avec un indice de 1_{10}).	
14b 14c 14d 14e 14f .... 15c 15d 15e 15f 1a2	01111011 (7b) 10000101 (85) 10000101 (85) 00001101 (0d) 01110011 (73) .... 01101110 (6e) 00001000 (08) 00111011 (3b) 01110101 (75) 01100100 (64)	Ces octets sont le codage d'un élément d'information **attribute** avec un nom qualifié littéral et une propriété **[normalized value]**. La présence des éléments d'information **attribute** était notée dans l'octet de position 05_{16} (le second bit est '1'). L'octet de position $14b_{16}$, valeur $7b_{16}$, a le premier bit à '0' (identification) notant qu'un élément d'information **attribute** est présent (voir § C.3.6.1). Les bits deux à cinq sont '1111' notant que le nom qualifié est un nom qualifié littéral (voir § C.17.3). Le bit six est '0' (bourrage) (voir § C.17.3). Le bit sept est '1' notant que le nom qualifié a une propriété **[prefix]**. Le bit huit est '1' notant que le nom qualifié a une propriété **[namespace name]**. L'octet de position $14c_{16}$, valeur 85_{16}, a '1' au premier bit notant que la propriété **[prefix]** n'est pas une chaîne littérale et qu'elle est indexée (voir § C.13.4). Le second bit est '0' notant que l'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 64_{10}, et l'indice est codé des bits trois à huit comme un entier non signé (voir§ C.25.2). L'entier non signé est 5_{10} et l'indice est 6_{10} (la limite inférieure est 1_{10}), ce qui donne une propriété **[prefix]** de "xsi" lors du déréférencement du tableau NAMESPACE NAME. L'octet de position $14d_{16}$, valeur 85_{16}, a '1' au premier bit notant que la propriété **[namespace name]** n'est pas une chaîne littérale et qu'elle est indexée (voir § C.13.4). Le second bit est '0' notant que l'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 64_{10}, et l'indice est codé des bits trois à huit comme un entier non signé (voir § C.25.2). L'entier non signé est 5_{10} et l'indice est 6_{10} (la limite inférieure est 1_{10}), ce qui donne une propriété **[namespace name]** de "….Schema-instance" lors du déréférencement du tableau NAMESPACE NAME.	**xsi:schemaLocation="…."**

Tableau D.10/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
		L'octet de position $14e_{16}$, valeur $0d_{16}$, a '0' au premier bit notant qu'une chaîne de caractères littérale est codée pour la propriété **[local name]** (voir § C.13.3). Le second bit est '0' notant que la longueur des caractères codés en UTF-8 est supérieure ou égale à 1_{10} et inférieure ou égale à 64_{10}, et la longueur est codée des bits trois à huit comme un entier non signé (voir § C.22.3.1). L'entier non signé est 13_{10} et la longueur est 14_{10} (la limite inférieure est 1_{10}). Les 14_{10} octets des caractères codés en UTF-8 (de la propriété **[local name]**) sont codés depuis l'octet de position $14f_{10}$ jusqu'à l'octet de position $15c_{10}$. La chaîne "SchemaLocation" sera ajoutée au tableau LOCAL NAME (avec un indice de 2_{10}). Le nom qualifié avec une propriété **[prefix]** de "xsi" (avec l'indice 6_{10}), une propriété **[namespace name]** de "….Schema-instance" (indice 6_{10}), et une propriété **[local name]** de "schemaLocation" (indice 2_{10}) sera ajouté au tableau ATTRIBUTE NAME (avec un indice de 1_{10}). L'octet de position $15d_{16}$, valeur 08_{16}, est le codage initial d'une chaîne ou indice non identifiant (voir § C.14) pour la propriété **[normalized value]**. Le premier bit est '0' notant qu'une chaîne de caractères littérale est présente (voir § C.14.3). Le second bit est '0' notant que la chaîne de caractères littérale ne devrait pas être ajoutée au tableau ATTRIBUTE VALUE. Les bits trois et quatre, tous deux à '0', notent que le format de codage de la chaîne est UTF-8 (voir § C.19.3.1). Les bits cinq et six sont respectivement à '1' et '0', notant que la longueur des octets des caractères UTF-8 (la propriété **[normalized value]**) est supérieure ou égale à 9_{10} octets et inférieure ou égale à 264_{10} octets, et que la longueur, moins la limite inférieure, est codée sur huit bits de l'octet suivant comme un entier non signé (voir § C.22.3.2). Les bits sept et huit sont à '0' (bourrage) (voir § C.22.3.2). L'octet de position $15e_{16}$, valeur $3b_{16}$, est le codage de l'entier non signé. La longueur des octets des caractères codés en UTF-8 est 68_{10} (la limite inférieure est 9_{10}). Les octets 68_{10} des caractères codés en UTF-8 (de la propriété **[normalized value]**) sont codés depuis l'octet de position $15f_{16}$ jusqu'à l'octet de position $1a2_{16}$.	
1a3	11110000 (f0)	Cet octet est le codage de la terminaison pour la séquence des éléments d'information **attribute**. L'octet de position $1a3_{16}$, valeur $f0_{16}$, a '1111' sur les quatre premiers bits (les bits un à quatre) et c'est la terminaison pour la séquence. Quatre '0' (bourrage) sont présents (les bits cinq à huit) si l'élément d'information **Order element** a des fils (voir § D.3.2).	

D.5.2.2 Codage de l'élément d'information Address element de l'élément d'information BuyerParty element

Les explications qui suivent détaillent le codage de l'élément d'information **Address element** de l'élément d'information **BuyerParty element** du document Fast Infoset. En particulier, le codage de l'élément d'information **element** et de l'élément d'information **character** est expliqué. Le Tableau D.11 présente le fragment du document Fast Infoset pour le codage de l'élément d'information **Address element** de l'élément d'information **BuyerParty element** du § D.3.2. Le Tableau D.12 détaille ce codage. Le fragment en XML 1.0 est présenté comme suit:

```
<cac:Address>
        <cbc:StreetName>Marsh Lane</cbc:StreetName>
        <cbc:CityName>Nowhere</cbc:CityName>
        <cbc:PostalZone>NR18 4XX</cbc:PostalZone>
        <cbc:CountrySubentity>Norfolk</cbc:CountrySubentity>
</cac:Address>
```

Tableau D.11/X.891 – Octets (comme caractères hexadécimaux) du fragment

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000200	3f838306416464726573
000220	733f828209537472656574e616d6582074d61727368204c616e65f03f828207
000240	436974794e616d6582044e6f7768657265f03f828209506f7374616c5a6f6e65
000260	82054e52313820345858f03f82820f436f756e747279537562656e7469747982
000280	044e6f72666f6c6bff

Tableau D.12/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
216 217 218 219 21a 220	00111111 (3f) 10000011 (83) 10000011 (83) 00000110 (06) 01000001 (41) 01110011 (73)	Ces octets sont le codage de l'élément d'information **Address element**. L'octet de position 216_{16}, valeur $3f_{16}$, a '0' (identification) au premier bit notant qu'il y a un fils d'un élément d'information **element** (fils de l'élément d'information **Party element**), et que le fils est un élément d'information **element** (voir § C.3.7.2). Le second bit est '0' notant que l'élément d'information **element** n'a pas d'attributs (voir § C.3.3). Les bits trois à cinq sont à '1111' notant que le nom qualifié est un nom qualifié littéral (voir § C.18.3). Le bit sept est à '1' notant que le nom qualifié a une propriété **[prefix]**. Le bit huit est à '1' notant que le nom qualifié a une propriété **[namespace name]**. L'octet de position 217_{16}, valeur 83_{16}, a '1' au premier bit notant que la propriété **[prefix]** n'est pas une chaîne littérale et qu'elle est indexée (voir § C.13.4). Le second bit est '0' notant que l'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 64_{10}, et l'indice est codé des bits trois à huit comme un entier non signé (voir § C.25.2). L'entier non signé est 3_{10} et l'indice est 4_{10} (la limite inférieure est 1_{10}), ce qui donne une propriété **[prefix]** de "cac" lors du déréférencement du tableau PREFIX. L'octet de position 218_{16}, valeur 83_{16}, a '1' au premier bit notant que la propriété **[namespace name]** n'est pas une chaîne littérale et qu'elle est indexée (voir § C.13.4). Le second bit est '0' notant que l'indice est supérieur ou égal à 1_{10} et inférieur ou égal à 64_{10}, et l'indice est codé des bits trois à huit comme un entier non signé (voir § C.25.2). L'entier non signé est 3_{10} et l'indice est 4_{10} (la limite inférieure est 1_{10}), ce qui donne une propriété **[namespace name]** de "….ateComponents:1:0" lors du déréférencement du tableau NAMESPACE NAME. L'octet de position 219_{16}, valeur 06_{16}, a '0' au premier bit notant qu'une chaîne de caractères littérale est codée pour la propriété **[local name]** (voir § C.13.3). Le second bit est '0' notant que la longueur des caractères codés en UTF-8 est supérieure ou égale à 1_{10} et inférieure ou égale à 64_{10}, et la longueur est codée du bit trois au bit huit comme un entier non signé (voir § C.22.3.1). L'entier non signé est 6_{10} et la longueur est 7_{10} (la limite inférieure est 1_{10}). Les 7_{10} octets des caractères codés en UTF-8 (de la propriété **[local name]**) sont codés depuis l'octet de position $21a_{16}$ jusqu'à l'octet de position 220_{16}. La chaîne "Address" sera ajoutée au tableau LOCAL NAME (avec un indice de 9_{10}). Le nom qualifié avec une propriété **[prefix]** de "cac" (indice 4_{10}), une propriété **[namespace name]** de "……..ateComponents:1:0" (indice 4_{10}), et une propriété **[local name]** de "Order" (indice 1_{10}) sera ajouté au tableau ELEMENT NAME (avec un indice de 1_{10}).	**<cac:Address>**

Tableau D.12/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
221 222 223 224 225 .... 22e	00111111 (3f) 10000010 (82) 10000010 (82) 00001001 (09) 01000001 (53) 01100101 (65)	Ces octets sont le codage de l'élément d'information **StreetName element**. La propriété **[local name]** "StreetName" sera ajoutée au tableau LOCAL NAME (avec un indice de 10_{10}). Le nom qualifié avec une propriété **[prefix]** de "cbc" (indice 3_{10}), une propriété **[namespace name]** de "…….BasicComponents:1:0" (indice 3_{10}), et une propriété **[local name]** de "StreetName" (indice 10_{10}) sera ajouté au tableau ELEMENT NAME (avec un indice de 9_{10}).	**<cbc:StreetName>**
22f 230 231 .... 23a	10000010 (82) 00000111 (07) 01001101 (4d) 01100101 (65)	Ces octets sont le codage des éléments d'information **character** de l'élément d'information **StreetName element**. L'octet de position $22f_{16}$, valeur 82_{16}, a '10' (identification) aux deux premiers bits (les bits un et deux) notant qu'il y a un fils d'un élément d'information **element** (fils de l'élément d'information **StreetName element**), et que le fils est un tronçon d'éléments d'information **character** (voir § C.3.7.5). Le troisième bit est '0' notant la présence d'une chaîne de caractères littérale (voir § C.15.3). Le bit quatre est à '0' notant que la chaîne de caractères littérale ne devrait pas être ajoutée au tableau CONTENT CHARACTER CHUNK. Les bits cinq et six, tous deux à '0', notent que le format de codage du tronçon est UTF-8 (voir § C.20.3.1). Les bits sept et huit sont respectivement à '1' et '0', notant que la longueur des octets des caractères codés en UTF-8 (le tronçon des éléments d'information **character**) est supérieure ou égale à 3_{10} octets et inférieure ou égale à 258_{10}, octets et que la longueur, moins la limite inférieure est codée en huit bits dans l'octet suivant comme un entier non signé (voir § C.24.3.2). L'octet de position 230_{16}, valeur 07_{16}, est l'entier non signalé. La longueur des octets des caractères codés en UTF-8 est 10_{10} (la limite inférieure est 3_{10}). Les 10_{10} octets des caractères codés en UTF-8 sont codés depuis l'octet de position 231_{16} jusqu'à l'octet de position $23a_{16}$.	Eléments d'information **character** "Marsh Lane"
23b	1111<u>0000</u> (f0)	Cet octet est la terminaison pour l'élément d'information **StreetName element**.	**</cbc:StreetName>**
23c 23d 23e 23f 240 247	00111111 (3f) 10000010 (82) 10000010 (82) 00000111 (07) 01000011 (43) 01100101 (65)	Ces octets sont le codage de l'élément d'information **CityName element**. La propriété **[local name]** "CityName" sera ajoutée au tableau LOCAL NAME (avec un indice de 10_{10}). Le nom qualifié avec une propriété **[prefix]** de "cbc" (indice 3_{10}), une propriété **[namespace name]** de "….BasicComponents:1:0" (indice 3_{10}), et une propriété **[local name]** de "CityName" (indice 11_{10}) sera ajouté au tableau ELEMENT NAME (avec un indice de 10_{10}).	**<cbc:CityName>**
248 249 24a 250	10000010 (82) 00000100 (04) 01001110 (4e) 01100101 (65)	Ces octets sont le codage des éléments d'information **character** de l'élément d'information **CityName element**. Les 7_{10} octets des caractères codés en UTF-8 sont codés depuis l'octet de position $24a_{16}$ jusqu'à l'octet de position 250_{16}.	Eléments d'information **character** "Nowhere"
251	1111<u>0000</u> (f0)	Cet octet est la terminaison de l'élément d'information **CityName element**.	**</cbc:CityName>**
252 253 254 255 256 .... 25f	00111111 (3f) 10000010 (82) 10000010 (82) 00001001 (09) 01000011 (50) 01100101 (65)	Ces octets sont le codage de l'élément d'information **PostalZone element**. La propriété **[local name]** "PostalZone" sera ajoutée au tableau LOCAL NAME (avec un indice de 12_{10}). Le nom qualifié avec la propriété **[prefix]** de "cbc" (indice 3_{10}), une propriété **[namespace name]** de "….BasicComponents:1:0" (indice 3_{10}), et une propriété **[local name]** de "PostalZone" (indice 12_{10}) sera ajouté au tableau ELEMENT NAME (avec un indice de 11_{10}).	**<cbc:PostalZone>**

Tableau D.12/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
260 261 262 269	10000010 (82) 00000101 (05) 01001110 (4e) 01011000 (58)	Ces octets sont le codage de l'élément d'information **character** de l'élément d'information **PostalZone element**. Les 8_{10} octets des caractères codés en UTF-8 sont codés depuis l'octet de position 262_{16} jusqu'à l'octet de position 269_{16}.	Eléments d'information **character** "NR18 4XX"
26a	1111<u>0000</u> (f0)	Cet octet est la terminaison de l'élément d'information **PostalZone element**.	**</cbc:PostalZone>**
26b 26c 26d 26e 26f 27e	00111111 (3f) 10000010 (82) 10000010 (82) 00001111 (0f) 01000011 (43) 01111001 (79)	Ces octets sont le codage de l'élément d'information **CountrySubentity element**. La propriété **[local name]** "CountrySubentity" sera ajoutée au tableau LOCAL NAME (avec un indice de 13_{10}). Le nom qualifié avec une propriété **[prefix]** de "cbc" (indice 3_{10}), une propriété **[namespace name]** de "….BasicComponents:1:0" (indice 3_{10}), et une propriété **[local name]** de "CountrySubentity" (indice 13_{10}) sera ajouté au tableau ELEMENT NAME (avec un indice de 12_{10}).	**<cbc:CountrySubentity>**
27f 280 281 287	10000010 (82) 00000100 (04) 01001110 (4e) 01101011 (6b)	Ces octets sont le codage des éléments d'information **character** de l'élément d'information **CountrySubentity element**. Les 7_{10} octets des caractères codés en UTF-8 sont codés depuis l'octet de position 281_{16} jusqu'à l'octet de position 287_{16}.	Eléments d'information **character** "Norfolk"
288	11111111 (ff)	Cet octet est la terminaison pour l'élément d'information **CountrySubentity element** et l'élément d'information **Address element**. L'octet de position 288_{16}, valeur ff_{16}, a '1111' (terminaison) au quatre premiers bits (les bits un à quatre) et c'est la terminaison pour l'élément d'information **CountrySubentity element** (voir § C.3.8). Les quatre derniers bits (du cinquième au huitième bit) sont à '1111' et c'est la terminaison pour l'élément d'information **Address element** (voir § C.3.8).	**</cbc:CountrySubentity>** **</cac:Address>**

D.5.2.3 Codage de l'élément d'information BuyersID element du premier élément d'information LineItem element

Les explications suivantes détaillent le codage de l'élément d'information **BuyersID element** du premier élément d'information **LineItem element** du document Fast Infoset. En particulier, est expliqué le codage d'un élément d'information **element** dont la propriété **[local name]** a été indexée avant que cet élément d'information ne soit expliqué. Le Tableau D.13 présente le fragment du document Fast Infoset pour le codage de l'élément d'information **BuyersID element** du premier élément d'information **LineItem element** du § D.14. Le Tableau D.14 détaille ce codage. Le fragment en XML 1.0 est présenté comme suit:

```
<cac:LineItem>
    <cac:BuyersID>A</cac:BuyersID>
```

Tableau D.13/X.891 – Octets (comme caractères hexadécimaux) du fragment

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
0003c0	<u>3f</u>8383074c696e654974656d<u>3f</u>8383829041<u>f0</u>

Tableau D.14/X.891 – Détails du codage

	Octet(s)	Description	XML infoset ou XML
3c4 3c5 3c6 3c7 3c8 3cf	00111111 (3f) 10000011 (83) 10000011 (83) 00001111 (07) 01001010 (4c) 01101101 (6d)	Cet octet est le codage de l'élément d'information **LineItem element**. La propriété **[local name]** "LineItem" sera ajoutée au tableau LOCAL NAME (avec un indice de 21_{10}). Le nom qualifié avec une propriété **[prefix]** de "cac" (indice 4_{10}), une propriété **[namespace name]** de "……..ateComponents:1:0" (indice 4_{10}), et une propriété **[local name]** de "LineItem" (indice 21_{10}) sera ajouté au tableau ELEMENT NAME (avec un indice de 20_{10}).	**<cac:LineItem>**
3d0 3d1 3d2 3d3	00111111 (3f) 10000011 (83) 10000011 (83) 10000010 (82)	Cet octet est le codage de l'élément d'information **BuyersID element**. La propriété **[prefix]** , la propriété **[namespace name]** et la propriété **[local name]** ont toutes été indexées car les chaînes associées sont toutes survenues avant cet élément d'information. La propriété **[local name]** a été indexée en traitant le premier fils de l'élément d'information **Order element**, à savoir l'élément d'information **BuyersID element** avec la propriété **[namespace name]** "….Order:1:0". Le nom qualifié avec une propriété **[prefix]** de "cac" (indice 4_{10}), une propriété **[namespace name]** de "……..ateComponents:1:0" (indice 4_{10}), et une propriété **[local name]** de "BuyersID" (indice 3_{10}) sera ajouté au tableau ELEMENT NAME (avec un indice de 21_{10}).	**<cac:BuyersID>**
3d4 3d5	10010000 (90) 01000001 (41)	Ces octets sont le codage de l'élément d'information **character** de l'élément d'information **BuyersID element**. L'octet de position $3d4_{16}$, valeur 90_{16}, a '10' (identification) aux deux premiers bits (les bits un et deux) notant qu'il y a un fils de l'élément d'information **element** (fils de l'élément d'information **BuyersID element**), et le fils est un tronçon des éléments d'information **character** (voir § C.3.7.5). Le troisième bit est '0' notant la présence d'une chaîne de caractères littérale (voir § C.15.3). Le quatrième bit est '1' notant que la chaîne de caractères littérale devrait être ajoutée au tableau CONTENT CHARACTER CHUNK (dans cet exemple des chaînes de moins de 6_{10} caractères sont ajoutées au tableau CONTENT CHARACTER CHUNK ou au tableau ATTRIBUTE VALUE). Les bits cinq et six, tous deux à '0', notent que le format de codage du tronçon est UTF-8 (voir § C.20.3.1). Le septième bit est à '0' notant que la longueur des octets des caractères codés en UTF-8 (le tronçon des éléments d'information **character**) est supérieur ou égal à l'octet 1_{10} et inférieur ou égal aux octets 2_{10}, et que la longueur, moins la limite inférieure, est codée dans le huitième bit comme un entier non signé (voir § C.24.3.1). L'entier non signé est 0_{10} et la longueur est 1_{10}. L'octet du caractère codé en UTF-8 est codé par l'octet de position 41_{16}.	Elément d'information **character** "A"
3d6	11110000 (f0)	Cet octet est la terminaison pour l'élément d'information **BuyersID element**.	**</cac:BuyersID>**

Annexe E

Allocation des valeurs d'identifiant d'objet

(La présente annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

Les identifiants d'objet et les descripteurs d'objet suivants sont alloués dans la présente Recommandation | Norme internationale:

{ joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) modules(0) fast-infoset(0)}

"Module ASN.1 Fast Infoset"

{joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) modules(0) fast-infoset-edm(1)}

"Module de définition de codage Fast Infoset"

{joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) modules(0) fast-infoset-elm(2)}

"Module de liaison de codage Fast Infoset"

{joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) codages(1) optional-xml-declaration(0)} -- *Defined as finf-doc-opt-decl in A.1*

"Document Fast Infoset contenant une déclaration XML"

{joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) codages(1) no-xml-declaration(1)} -- *Defined as finf-doc-no-decl in A.1*

"Document Fast Infoset sans déclaration XML"

BIBLIOGRAPHIE

[1] OASIS *Universal Business Language (UBL) 1.0* (Langage d'affaires universel).

[2] IETF RFC 1952 (1996), *GZIP file format specification version 4.3* (Spécification du format de fichier GZIP).

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série D	Principes généraux de tarification
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	Gestion des télécommunications y compris le RGT et maintenance des réseaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	Commutation et signalisation
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	**Réseaux de données, communication entre systèmes ouverts et sécurité**
Série Y	Infrastructure mondiale de l'information, protocole Internet et réseaux de prochaine génération
Série Z	Langages et aspects généraux logiciels des systèmes de télécommunication

Imprimé en Suisse
Genève, 2007