Cahiers GUTenberg

LA SPÉCIFICATION XML 1.0

CONSORTIUM WWW

Cahiers GUTenberg, nº 33-34 (1999), p. 191-280.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1999___33-34_191_0>

La spécification XML

Consortium WWW

W3C, MIT, INRIA, Keio

Cette annexe contient la version originale en anglais (en pages impaires) et la traduction française (en pages paires) de la recommandation *Extensible Markup Language* (XML) 1.0 (`http://www.w3.org/TR/1998/REC-xml-19980210`) du W3C, datée du 10 février 1998.

La version traduite peut contenir des erreurs absentes de l'original, dues à la traduction elle-même. Ainsi la version originale en anglais à la page de droite est-elle la seule normative.

Les personnes suivantes ont participé à la traduction :

— Patrick ANDRIES (CAFI), courriel : `<pandries@cafi.org>` ;
— Samira CUNY (Université de Bristol), courriel `<glsc@bris.ac.uk>` ;
— Alain LA BONTÉ (Guébec), courriel `<alb@sct.gouv.qc.ca>` ;
— Nicolas LESBATS, courriel `<nlesbats@hotmail.com>` ;
— François YERGEAU (Alis Technologies), courriel : `<yergeau@alis.com>`.

Le fichier HTML de la traduction ainsi que quelques autres informations au sujet du travail original de ces personnes est disponible à l'URL `http://babel.alis.com/web_ml/xml/`. L'introduction à XML de ce *Cahier* y a déjà fait référence (annexe F : « Terminologie bilingue XML » en page 115). On trouvera plus de détails sur cette traduction à la page 280.

Copyright

Langage de balisage extensible (XML) 1.0

Recommandation du W3C, 10 février 1998

Cette version :
`http://www.w3.org/TR/1998/REC-xml-19980210`
`http://www.w3.org/TR/1998/REC-xml-19980210.xml`
`http://www.w3.org/TR/1998/REC-xml-19980210.html`
`http://www.w3.org/TR/1998/REC-xml-19980210.pdf`
`http://www.w3.org/TR/1998/REC-xml-19980210.ps`
Version la plus récente :
`http://www.w3.org/TR/REC-xml`
Version précédente :
`http://www.w3.org/TR/PR-xml-971208`
Rédacteurs :
Tim Bray (Textuality and Netscape) `<tbray@textuality.com>`
Jean Paoli (Microsoft) `<jeanpa@microsoft.com>`
C. M. Sperberg-McQueen (University of Illinois at Chicago) `<cmsmcq@uic.edu>`.

Sommaire

Le langage de balisage extensible (Extensible Markup Language, XML) est un sous-ensemble de SGML qui est complètement décrit dans ce document. Son but est de permettre au SGML générique d'être transmis, reçu et traité sur le Web de la même manière que l'est HTML aujourd'hui. XML a été conçu pour être facile à mettre en œuvre et interopérable avec SGML et HTML.

Statut de ce document

Ce document a été examiné par des membres du W3C et d'autres parties intéressées et a été approuvé par le Directeur comme Recommandation du W3C. Ce document est stable et peut servir de référence ou être cité comme standard dans un autre document. En promulgant cette recommandation, le W3C cherche à attirer l'attention sur la spécification et à promouvoir sa mise en œuvre. Cette action améliore la fonctionalité et l'interopérabilité du Web.

Extensible Markup Language (XML) 1.0

W3C Recommendation 10-February-1998

This version:

`http://www.w3.org/TR/1998/REC-xml-19980210`
`http://www.w3.org/TR/1998/REC-xml-19980210.xml`
`http://www.w3.org/TR/1998/REC-xml-19980210.html`
`http://www.w3.org/TR/1998/REC-xml-19980210.pdf`
`http://www.w3.org/TR/1998/REC-xml-19980210.ps`

Latest version:

`http://www.w3.org/TR/REC-xml`

Previous version:

`http://www.w3.org/TR/PR-xml-971208`

Editors:

Tim Bray (Textuality and Netscape) `<tbray@textuality.com>`
Jean Paoli (Microsoft) `<jeanpa@microsoft.com>`
C. M. Sperberg-McQueen (University of Illinois at Chicago) `<cmsmcq@uic.edu>`.

Abstract

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Ce document précise une syntaxe créée en extrayant un sous-ensemble d'une norme internationale de traitement de texte existante et largement utilisée (le langage normalisé de balisage généralisé, ISO 8879:1986(F) tel qu'amendé et corrigé), dans le but de l'utiliser sur le Web. C'est un produit de l'activité XML du W3C, sur laquelle on trouvera de l'information à l'adresse `http://www.w3.org/XML`. Une liste des recommandations en vigueur du W3C et d'autres documents techniques se trouve à l'adresse `http://www.w3.org/TR`.

Ce document utilise le terme URI, défini dans [Berners-Lee et al.], un travail en cours destiné à mettre à jour les documents [IETF RFC1738] et [IETF RFC1808].

La liste des erreurs connues de cette spécification est disponible à l'adresse `http://www.w3.org/XML/xml-19980210-errata`.

On est prié de signaler toute erreur dans ce document à `<xml-editor@w3.org>`.

Langage de balisage extensible (XML) 1.0

Table des matières

This document specifies a syntax created by subsetting an existing, widely used international text processing standard (Standard Generalized Markup Language, ISO 8879:1986(E) as amended and corrected) for use on the World Wide Web. It is a product of the W3C XML Activity, details of which can be found at `http://www.w3.org/XML`. A list of current W3C Recommendations and other technical documents can be found at `http://www.w3.org/TR`.

This specification uses the term URI, which is defined by [Berners-Lee et al.], a work in progress expected to update [IETF RFC1738] and [IETF RFC1808].

The list of known errors in this specification is available at

`http://www.w3.org/XML/xml-19980210-errata`.

Please report errors in this document to `<xml-editor@w3.org>`.

Extensible Markup Language (XML) 1.0

Table of Contents

Annexes

Appendices

1. Introduction

Le Langage de balisage extensible [en anglais Extensible Markup Language] (abrégé XML) décrit une classe d'objets de données appelés documents XML et décrit partiellement le comportement des programmes qui les traitent. XML est un profil d'application ou une forme restreinte de SGML, le langage normalisé de balisage généralisé [ISO 8879]. Par construction, les documents XML sont des documents conformes à SGML.

Les documents XML se composent d'unités de stockage appelées entités, qui contiennent des données analysables ou non. Les données analysables se composent de caractères, certains formant les données textuelles, et le reste formant le balisage. Le balisage décrit les structures logique et de stockage du document. XML fournit un mécanisme pour imposer des contraintes à ces structures.

Un module logiciel appelé **processeur XML** est utilisé pour lire les documents XML et pour accéder à leur contenu et à leur structure. On suppose qu'un processeur XML effectue son travail pour le compte d'un autre module, appelé **l'application**. Cette spécification décrit le comportement requis d'un processeur XML, c'est à dire la manière dont il doit lire des données XML et les informations qu'il doit fournir à l'application.

1.1 Origine et buts

XML a été développé par un groupe de travail (GT) XML [XML Working Group] (initialement connu sous le nom de comité d'examen éditorial SGML [SGML Editorial Review Board]) constitué sous les auspices du Consortium du World Wide Web (W3C) en 1996. Le GT était présidé par Jon Bosak de Sun Microsystems avec la participation active d'un groupe d'intérêt XML [XML Special Interest Group] (auparavant connu sous le nom de groupe de travail de SGML [SGML Working Group]) également organisé par le W3C. La liste des membres du groupe de travail XML est donnée en annexe. Dan Connolly agissait comme contact du W3C auprès du GT.

Les objectifs de conception de XML sont les suivants :

1. XML devrait pouvoir être utilisé sans difficulté sur Internet ;
2. XML devrait soutenir une grande variété d'applications ;
3. XML devra être compatible avec SGML ;
4. Il devrait être facile d'écrire des programmes traitant les documents XML ;
5. Le nombre d'options dans XML doit être réduit au minimum, idéalement à aucune ;
6. Les documents XML devraient être lisibles par l'homme et raisonnablement clairs ;
7. La conception de XML devrait être préparée rapidement ;
8. La conception de XML sera formelle et concise ;
9. Il devrait être facile de créer des documents XML ;
10. La concision dans le balisage de XML est de peu d'importance.

1. Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an **XML processor** is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the **application**. This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. The membership of the XML Working Group is given in an appendix. Dan Connolly served as the WG's contact with the W3C.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

Cette spécification, ainsi que certaines normes associées (Unicode et l'ISO/CEI 10646 pour les caractères, le RFC 1766 Internet pour les balises d'identification de langue, l'ISO 639 pour les codes de noms de langue et l'ISO 3166 pour les codes de noms de pays) fournissent toutes les informations nécessaires pour comprendre la version 1.0 de XML et pour construire des programmes pour la traiter.

Cette version de la spécification de XML peut être distribuée librement, à condition que tout le texte et les notices juridiques demeurent intacts.

1.2 Terminologie

La terminologie employée pour décrire les documents XML est définie dans cette spécification. Les termes définis dans la liste suivante sont utilisés pour établir ces définitions et pour décrire les actions d'un processeur XML :

pouvoir (verbe) il n'y a pas obligation pour les documents conformes et les processeurs XML de se comporter tel que décrit.

devoir (verbe) il n'y a pas obligation pour les documents conformes et les processeurs XML de se comporter tel que décrit ; tout manquement est une erreur.

erreur une violation des règles de cette spécification. Les résultats sont indéterminés. Un logiciel conforme peut détecter et signaler une erreur et peut s'en remettre.

erreur fatale une erreur qu'un processeur XML conforme doit détecter et signaler à l'application. À la suite d'une erreur fatale, le processeur peut continuer à traiter les données pour rechercher d'autres erreurs et peut signaler de telles erreurs à l'application. Afin d'aider la correction des erreurs, le processeur peut rendre disponibles à l'application des données non-traitées (mélange de données textuelles et de balisage). Dès qu'une erreur fatale est détectée, toutefois, le processeur ne doit pas continuer le traitement normal (c'est à dire qu'il ne doit pas continuer à transmettre à l'application d'une façon normale les données textuelles et l'information sur la structure logique du document).

au gré de l'utilisateur un logiciel conforme peut ou doit (selon le verbe dans la phrase) se comporter tel que décrit ; s'il le fait, il doit fournir à l'utilisateur un moyen d'activer ou de désactiver le comportement décrit.

contrainte de validité une règle qui s'applique à tous les documents XML valides. Les violations des contraintes de validité sont des erreurs ; elles doivent, au gré de l'utilisateur, être signalées par les processeurs XML validateurs.

contrainte de forme une règle qui s'applique à tous les documents XML bien formés. Les violations des contraintes de forme sont des erreurs fatales.

correspondance (De chaînes de caractères ou de noms :) Deux chaînes de caractères ou deux noms correspondent s'ils sont identiques. Les caractères possédant des représentations multiples dans l'ISO/CEI 10646 (c-à-d les caractères avec des formes précomposée et base+diacritiques) correspondent seulement s'ils ont la même représentation dans les deux chaînes de caractères. Au gré de l'utilisateur, les processeurs peuvent normaliser de tels caractères à une certaine forme canonique. Aucune transformation de casse n'est effectuée. (De chaînes de caractères et de règles de grammaire :) Toute chaîne appartenant au langage engendré par une production grammaticale est réputée correspondre à cette production. (Du contenu et des modèles de contenu :) un élément correspond à sa déclaration quand il se conforme à la forme décrite dans la contrainte « élément valide ».

This specification, together with associated standards (Unicode and ISO/IEC 10646 for characters, Internet RFC 1766 for language identification tags, ISO 639 for language name codes, and ISO 3166 for country name codes), provides all the information necessary to understand XML Version 1.0 and construct computer programs to process it.

This version of the XML specification may be distributed freely, as long as all text and legal notices remain intact.

1.2 Terminology

The terminology used to describe XML documents is defined in the body of this specification. The terms defined in the following list are used in building those definitions and in describing the actions of an XML processor:

may Conforming documents and XML processors are permitted to but need not behave as described.

must Conforming documents and XML processors are required to behave as described; otherwise they are in error.

error A violation of the rules of this specification; results are undefined. Conforming software may detect and report an error and may recover from it.

fatal error An error which a conforming XML processor must detect and report to the application. After encountering a fatal error, the processor may continue processing the data to search for further errors and may report such errors to the application. In order to support correction of errors, the processor may make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor must not continue normal processing (i.e., it must not continue to pass character data and information about the document's logical structure to the application in the normal way).

at user option Conforming software may or must (depending on the modal verb in the sentence) behave as described; if it does, it must provide users a means to enable or disable the behavior described.

validity constraint A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must, at user option, be reported by validating XML processors.

well-formedness constraint A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are fatal errors.

match (Of strings or names:) Two strings or names being compared must be identical. Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. At user option, processors may normalize such characters to some canonical form. No case folding is performed. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production. (Of content and content models:) An element matches its declaration when it conforms in the fashion described in the constraint "Element Valid".

à des fins de compatibilité un dispositif de XML uniquement inclus pour s'assurer que XML demeure compatible avec SGML.

à des fins d'interopérabilité une recommandation non-contraignante incluse pour accroître les chances de traitement de document XML par des processeurs SGML antérieurs à l'annexe « WebSGML Adaptations » de l'ISO 8879.

2. Documents

Un objet de données est un **document XML** s'il est bien formé, tel que précisé dans cette spécification. De plus, un document XML bien formé peut être valide s'il obéit à certaines autres contraintes.

Chaque document XML a une structure logique et une structure physique. Physiquement, le document se compose d'unités appelées entités. Une entité peut appeler d'autres entités pour causer leur inclusion dans le document. Un document commence à la « racine » ou entité document. Logiquement, le document se compose de déclarations, d'éléments, de commentaires, d'appels de caractère et d'instructions de traitement, qui sont indiqués dans le document par du balisage explicite. Les structures logiques et physiques doivent s'imbriquer correctement, tel que décrit dans « 4.3.2 Entités analysables bien formées ».

2.1 Documents XML bien formés

Un objet textuel est un document XML bien formé si :

1. pris dans son ensemble, il correspond à la production étiquetée `document` ;
2. il obéit à toutes les contraintes de forme données dans cette spécification ;
3. chacune des entités analysables qui est appelée directement ou indirectement dans le document est bien formée.

Document

```
[1] document ::= prologue élément Divers*
```

La correspondance à la production `document` implique que :

1. le document contient un ou plusieurs éléments ;
2. il y a un seul élément, appelé la **racine**, ou l'élément document, dont aucune partie n'apparaît dans le contenu d'un autre élément. Pour tous les autres éléments, si la balise de début est dans le contenu d'un autre élément, la balise de fin est dans le contenu du même élément. Autrement dit, les éléments, délimités par des balises de départ et de fin, s'imbriquent correctement les uns dans les autres.

En conséquence, pour chaque élément `F` (autre que la racine) dans le document, il y a un autre élément `P` dans le document tel que `F` est dans le contenu de `P`, mais n'est dans le contenu d'aucun autre élément qui est dans le contenu de `P`. `P` est connu comme **parent** de `F`, et `F` comme **fils** de `P`.

for compatibility A feature of XML included solely to ensure that XML remains compatible with SGML.

for interoperability A non-binding recommendation included to increase the chances that XML documents can be processed by the existing installed base of SGML processors which predate the WebSGML Adaptations Annex to ISO 8879.

2. Documents

A data object is an **XML document** if it is well-formed, as defined in this specification. A well-formed XML document may in addition be valid if it meets certain further constraints.

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures must nest properly, as described in "4.3.2 Well-Formed Parsed Entities".

2.1 Well-Formed XML Documents

A textual object is a well-formed XML document if:

1. Taken as a whole, it matches the production labeled `document`.
2. It meets all the well-formedness constraints given in this specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

Document

```
[1] document ::= prolog element Misc*
```

Matching the `document` production implies that:

1. It contains one or more elements.
2. There is exactly one element, called the **root**, or document element, no part of which appears in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

As a consequence of this, for each non-root element `C` in the document, there is one other element `P` in the document such that `C` is in the content of `P`, but is not in the content of any other element that is in the content of `P`. `P` is referred to as the **parent** of `C`, and `C` as a **child** of `P`.

2.2 Caractères

Une entité analysable contient du **texte**, une suite de caractères qui peuvent représenter du balisage ou des données textuelles. Un **caractère** est une unité de texte tel que précisé par l'ISO/CEI 10646 [ISO/CEI 10646]. Les caractères admissibles sont la tabulation, le retour de chariot, le retour à la ligne, et les caractères graphiques d'Unicode et de l'ISO/CEI 10646. L'utilisation des « caractères de compatibilité », tel que définis dans la section 6.8 de [Unicode], est déconseillée.

Ensemble de caractères

```
[2] Car ::= #x9 | #xA | #xD | [#x20-#xD7FF] |
            [#xE000-#xFFFD] | [#x10000-#x10FFFF]
              /* tout caractère Unicode, sauf les seizets d'indirection,
                                                          FFFE et FFFF. */
```

Le mécanisme de codage des positions de code de caractère en configurations binaires peut varier d'une entité à l'autre. Tous les processeurs XML doivent accepter les codages UTF-8 et UTF-16 de l'ISO/CEI 10646 ; les mécanismes pour signaler lequel des deux est utilisé ou pour introduire d'autres codages sont discutés ci-dessous dans « 4.3.3 Codage des caractères dans les entités ».

2.3 Constructions syntaxiques communes

Ce paragraphe définit quelques symboles souvent utilisés dans la grammaire.

`S` (séparateurs, blanc) se compose d'un ou plusieurs des caractères espace (#x20), retour de chariot, retour à la ligne, ou tabulation.

Séparateurs

```
[3] S ::= (#x20 | #x9 | #xD | #xA)+
```

Par commodité, les caractères sont classifiés en lettres, chiffres ou autres caractères. Les lettres se composent d'un caractère de base alphabétique ou syllabique, éventuellement suivi d'un ou plusieurs caractères combinatoire, ou encore d'un caractère idéographique. Des définitions complètes des classes de caractères sont données dans « B. Classes de caractères ».

Un **nom** est une unité lexicale commençant par une lettre ou par un des caractères de ponctuation d'une courte liste, suivi de lettres, de chiffres, de traits d'union, de traits de soulignement, de deux points ou de points finals, tous connus comme caractères constitutifs de nom. Les noms commençant par la chaîne de caractères « `xml` », ou par n'importe quelle chaîne de caractères correspondant à `(('X'|'x') ('M'|'m') ('L'|'l'))`, sont réservés à des fins de normalisation dans cette version ou dans des versions ultérieures de la spécification.

2.2 Characters

A parsed entity contains **text**, a sequence of characters, which may represent markup or character data. A **character** is an atomic unit of text as specified by ISO/IEC 10646 [ISO/IEC 10646]. Legal characters are tab, carriage return, line feed, and the legal graphic characters of Unicode and ISO/IEC 10646. The use of "compatibility characters", as defined in section 6.8 of [Unicode], is discouraged.

Character Range

```
[2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF] |
             [#xE000-#xFFFD] | [#x10000-#x10FFFF]
                /*any Unicode character, excluding the surrogate blocks,
                                                        FFFE, and FFFF. */
```

The mechanism for encoding character code points into bit patterns may vary from entity to entity. All XML processors must accept the UTF-8 and UTF-16 encodings of 10646; the mechanisms for signaling which of the two is in use, or for bringing other encodings into play, are discussed later, in "4.3.3 Character Encoding in Entities".

2.3 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tabs.

White Space

```
[3] S ::= (#x20 | #x9 | #xD | #xA)+
```

Characters are classified for convenience as letters, digits, or other characters. Letters consist of an alphabetic or syllabic base character possibly followed by one or more combining characters, or of an ideographic character. Full definitions of the specific characters in each class are given in "B. Character Classes".

A **Name** is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters. Names beginning with the string "`xml`", or any string which would match `(('X'|'x') ('M'|'m') ('L'|'l'))`, are reserved for standardization in this or future versions of this specification.

Note : Le caractère deux points dans les noms XML est réservé pour l'expérimentation avec les espaces de nom. On s'attend à ce que sa signification soit normalisée bientôt ; les documents utilisant les deux points dans un but expérimental pourront alors nécessiter une mise à jour. (Il n'y a aucune garantie que le mécanisme d'espace de nom éventuellement adopté pour XML utilise en fait les deux points comme séparateur.) Dans la pratique, ceci signifie que les auteurs ne devraient pas utiliser les deux points dans les noms XML, sauf en tant qu'élément expérimental d'espace de nom, mais que les processeurs XML devraient accepter les deux points comme caractère constitutif de nom.

`Un AtomeNml` (atome nominal) est une suite de caractères constitutifs de nom.

Noms et atomes nominaux

```
[4]     CarNom ::= Lettre | Chiffre | '.'  | '-' | '_' | ':'  |
                   CarJonctif |  ModificateurLettre
[5]        Nom ::= (Lettre | '_' | ':')  (CarNom)*
[6]       Noms ::= Nom (S Nom)*
[7]  AtomeNml ::= (CarNom)+
[8] AtomesNmx ::= AtomeNml (S AtomeNml)*
```

Une chaîne délimitée, ne contenant pas les guillemets (anglais, " ou ') utilisés comme délimiteur, constitue ce que l'on appelle « un littéral ». Des littéraux sont utilisés pour préciser le contenu des entités internes (`ValeurEntité`), des valeurs des attributs (`ValeurAtt`) et des identificateurs externes (`LittéralSystème`). Notez qu'un `LittéralSystème` peut être analysé sans rechercher le balisage.

Littéraux

```
 [9]     ValeurEntité ::= '"' ([^%&"] | AppelEP | Appel)* '"' |
                          "'" ([^%&'] | AppelEP | Appel)*"'"
[10]        ValeurAtt ::= '"' ([^<&"] | Appel)* '"' |
                          "'" ([^<&'] | Appel)* "'"
[11] LittéralSystème ::= ('"' [^"]* '"') |("'" [^']* "'")
[12]   IdPubLittéral ::= '"' CarIdPub* '"' |
                          "'" (CarIdPub -"'")* "'"
[13]         CarIdPub ::= #x20 | #xD | #xA | [a-zA-Z0-9] |
                          [-'()+,./:=?;!*#@$_%]
```

2.4 Données textuelles et balisage

Le texte se compose de données textuelles et de balisage. Le **balisage** prend la forme de balises ouvrantes, de balises fermantes, de balises d'éléments vides, d'appels d'entité, d'appels de caractère, de commentaires, de délimiteurs de section CDATA, de déclarations de type de document, et d'instructions de traitement.

Tout le texte qui n'est pas du balisage constitue les **données textuelles** du document.

Note: The colon character within XML names is reserved for experimentation with name spaces. Its meaning is expected to be standardized at some future point, at which point those documents using the colon for experimental purposes may need to be updated. (There is no guarantee that any name-space mechanism adopted for XML will in fact use the colon as a name-space delimiter.) In practice, this means that authors should not use the colon in XML names except as part of name-space experiments, but that XML processors should accept the colon as a name character.

An `Nmtoken` (name token) is any mixture of name characters.

Names and Tokens

```
[4] NameChar ::= Letter | Digit | '.'  | '-' | '_' | ':'  |
                 CombiningChar |  Extender
[5]     Name ::= (Letter | '_' | ':')  (NameChar)*
[6]    Names ::= Name (S Name)*
[7]  Nmtoken ::= (NameChar)+
[8] Nmtokens ::= Nmtoken (S Nmtoken)*
```

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string. Literals are used for specifying the content of internal entities (`EntityValue`), the values of attributes (`AttValue`), and external identifiers (`SystemLiteral`). Note that a `SystemLiteral` can be parsed without scanning for markup.

Literals

```
 [9]   EntityValue ::= '"' ([^%&"] | PEReference | Reference)* '"' |
                       "'" ([^%&'] | PEReference | Reference)*"'"
[10]      AttValue ::= '"' ([^<&"] | Reference)* '"' |
                       "'" ([^<&'] | Reference)* "'"
[11] SystemLiteral ::= ('"' [^"]* '"') |("'" [^']* "'")
[12]  PubidLiteral ::= '"' PubidChar* '"' |
                        "'" (PubidChar -"'")* "'"
[13]     PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9] |
                       [-'()+,./:=?;!*#@$_%]
```

2.4 Character Data and Markup

Text consists of intermingled character data and markup. **Markup** takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, and processing instructions.

All text that is not markup constitutes the **character data** of the document.

Les caractères esperluète (&) et crochet gauche (<) peuvent apparaître sous leur forme littérale *seulement* quand ils sont utilisés comme délimiteurs de balisage ou dans un commentaire, une instruction de traitement, ou une section CDATA. Ils sont également permis dans la valeur littérale d'une entité dans une déclaration d'entité interne ; voir « 4.3.2 Entités analysables bien formées ». S'ils sont nécessaires ailleurs, ils doivent être déguisés en utilisant des appels de caractères numériques ou en utilisant les chaînes de caractères « `&` » et « `<` » respectivement. Le crochet droit (>) peut être représenté en utilisant la chaîne de caractères « `>` » et doit, pour compatibilité, être déguisé en utilisant « `>` » ou un appel de caractère quand il apparaît dans la chaîne de caractères « `]]>` » dans du contenu, quand cette chaîne ne marque pas la fin d'une section CDATA.

Dans le contenu des éléments, toute chaîne de caractères ne contenant pas un délimiteur de début de balisage est considéré donnée textuelle. Dans une section CDATA, toute chaîne de caractères ne contenant pas le délimiteur de fin de section CDATA, « `]]>` », est considéré donnée textuelle.

L'apostrophe (') peut être représentée par « `&apos;` », et le caractère guillemet anglais (") par « `&quot;` », afin de permettre à des valeurs d'attribut de contenir ces caractères.

Données textuelles

```
[14] DonnéesTextuelles ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5 Commentaires

Les **commentaires** peuvent apparaître n'importe où dans un document en dehors d'autre balisage ; de plus, ils peuvent apparaître dans la déclaration de type de document aux endroits permis par la grammaire. Ils ne font pas partie des données textuelles du document ; un processeur XML peut permettre à une application de récupérer le texte des commentaires. À des fins de compatibilité, la chaîne « - » (double trait d'union) ne doit pas apparaître à l'intérieur de commentaires.

Commentaires

```
[15] Commentaires ::= '<!-' ((Car - '-') | ('-' (Car - '-')))* '->'
```

Exemple de commentaire :

```
<!- déclarations pour <en-tête> & <corps> ->
```

2.6 Instructions de traitement

Les **instructions de traitement** (IT) permettent aux documents de contenir des instructions pour des applications.

The ampersand character (&) and the left angle bracket (<) may appear in their literal form *only* when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section. They are also legal within the literal entity value of an internal entity declaration; see "4.3.2 Well-Formed Parsed Entities". If they are needed elsewhere, they must be escaped using either numeric character references or the strings "`&`" and "`<`" respectively. The right angle bracket (>) may be represented using the string "`>`", and must, for compatibility, be escaped using "`>`" or a character reference when it appears in the string "`]]>`" in content, when that string is not marking the end of a CDATA section.

In the content of elements, character data is any string of characters which does not contain the start-delimiter of any markup. In a CDATA section, character data is any string of characters not including the CDATA-section-close delimiter, "`]]>`".

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "`&apos;`", and the double-quote character (") as "`&quot;`".

Character Data

```
[14] CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5 Comments

Comments may appear anywhere in a document outside other markup; in addition, they may appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "`-`" (double-hyphen) must not occur within comments.

Comments

```
[15] Comment ::= '<!-' ((Char - '-') | ('-' (Char - '-')))* '->'
```

An example of a comment:

```
<!- declarations for <head> & <body> ->
```

2.6 Processing Instructions

Processing instructions (PIs) allow documents to contain instructions for applications.

Instructions de traitement

```
[16]      IT ::= '<?'  CibleIT (S (Car'* - (Car* '?>' Car*)))? '?>'
[17] CibleIT ::= Nom - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

Les IT ne font pas partie des données textuelles des documents mais doivent être transmises à l'application. Une IT commence par une cible (`CibleIT`) qui identifie l'application à laquelle l'instruction est destinée. Les noms de cible « `XML` », « `xml` » et ainsi de suite sont réservés à des fins de standardisation dans cette version ou des versions ultérieures de cette spécification. Le mécanisme de Notation de XML peut être utilisé pour la déclaration formelle des cibles d'IT.

2.7 Sections CDATA

Les **sections CDATA** peuvent se trouver à n'importe quel endroit acceptable pour des données textuelles ; elles sont employées pour déguiser des blocs de texte contenant des caractères qui seraient autrement identifiés comme balisage. Les sections CDATA commencent par la chaîne « `<![CDATA[` » et se terminent par la chaîne « `]]>` ».

Sections CDATA

```
[18] SectionDT ::= DébutDT DonnéesDT FinDT
[19]   DébutDT ::= '<![CDATA['
[20] DonnéesDT ::= (Car* - (Car* ']]>' Car*))
[21]     FinDT ::= ']]>'
```

Dans une section CDATA, seule la chaîne de caractères `FinDT` est identifiée comme balisage, de sorte que les crochets gauches et les esperluètes peuvent s'y trouver littéralement ; ils n'ont pas besoin (et ne peuvent pas) être déguisés en utilisant « `<` » et « `&` ». Les sections CDATA ne peuvent pas s'imbriquer.

Voici un exemple de section CDATA, dans lequel « `<accueil>` » et « `</accueil>` » sont reconnus comme données textuelles, et non comme balisage :

```
<![CDATA[<accueil>Bonjour!</accueil>]]>
```

2.8 Prologue et déclaration de type de document

Les documents XML peuvent, et devraient, commencer par une **déclaration XML** qui indique la version de XML utilisée. L'exemple suivant est un document XML, bien formé mais non valide :

```
<?xml~version="1.0"?>
<accueil>Bonjour!</accueil>
```

ainsi que celui-ci :

```
<accueil>Bonjour!</accueil>
```

Processing Instructions

```
[16]       PI ::= '<?'  PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

PIs are not part of the document's character data, but must be passed through to the application. The PI begins with a target (`PITarget`) used to identify the application to which the instruction is directed. The target names "`XML`", "`xml`", and so on are reserved for standardization in this or future versions of this specification. The XML Notation mechanism may be used for formal declaration of PI targets.

2.7 CDATA Sections

CDATA sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "`<![CDATA[`" and end with the string "`]]>`":

CDATA Sections

```
[18]  CDSect ::= CDStart CData CDEnd
[19] CDStart ::= '<![CDATA['
[20]   CData ::= (Char* - (Char* ']]>' Char*))
[21]   CDEnd ::= ']]>'
```

Within a CDATA section, only the `CDEnd` string is recognized as markup, so that left angle brackets and ampersands may occur in their literal form; they need not (and cannot) be escaped using "`<`" and "`&`". CDATA sections cannot nest.

An example of a CDATA section, in which "`<greeting>`" and "`</greeting>`" are recognized as character data, not markup:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8 Prolog and Document Type Declaration

XML documents may, and should, begin with an **XML declaration** which specifies the version of XML being used. For example, the following is a complete XML document, well-formed but not valid:

```
<?xml~version="1.0"?>
<greeting>Hello, world!</greeting>
```

and so is this:

```
<greeting>Hello, world!</greeting>
```

Le numéro de version « 1.0 » devrait être employé pour indiquer la conformité à cette version de la spécification ; un document utilisant la valeur « 1.0 » mais ne se conformant pas à cette version de la spécification est en erreur. Le Groupe de travail XML a l'intention de produire des versions ultérieures à la version « 1.0 », mais cette intention ne signifie aucunement un engagement à produire de futures versions de XML ni, si d'autres versions sont produites, à utiliser un plan de numérotation particulier. Puisque de futures versions ne sont pas exclues, cette construction est un moyen de permettre l'identification automatique de la version, si celle-ci devient nécessaire. Les processeurs peuvent signaler une erreur s'ils reçoivent des documents étiquetés avec des versions qu'ils ne connaissent pas.

La fonction du balisage dans un document XML est de décrire sa structure de stockage et sa structure logique et d'associer des paires attributs-valeurs à ses structures logiques. XML fournit un mécanisme, la déclaration de type de document, pour définir des contraintes sur la structure logique et pour gérer l'utilisation des unités de stockage prédéfinies.Un document XML est **valide** si une déclaration de type de document y est associée et si le document est conforme aux contraintes qu'elle exprime.

La déclaration de type de document doit apparaître avant le premier élément dans le document.

Prologue

```
[22]    prologue ::= DéclXML? Divers* (déclTypeDoc Divers*)?
[23]     DéclXML ::= '<?xml' InfoVersion DéclCodage? DéclDocAuto? S? '?>'
[24] InfoVersion ::= S 'version' Égal (' NumVersion ' | " NumVersion ")
[25]       Égal ::= S? '=' S?
[26]  NumVersion ::= ([a-zA-Z0-9_.:]  | '-')+
[27]      Divers ::= Commentaires| IT |  S
```

La **déclaration de type de document** XML contient ou désigne des déclarations de balisage qui fournissent une grammaire pour une classe de documents. Cette grammaire est connue comme définition de type de document, ou **DTD**. La déclaration de type de document peut désigner un sous-ensemble externe (un genre spécial d'entités externes) contenant des déclarations de balisage, peut contenir des déclarations de balisage directement dans un sous-ensemble interne ou peut faire les deux. La DTD d'un document se compose des deux sous-ensembles regroupés.

Une **déclaration de balisage** est une déclaration de type d'élément, une déclaration de liste d'attributs, une déclaration d'entités ou une déclaration de notation. Ces déclarations peuvent être contenues entièrement ou partiellement dans des entités paramètres, tel que décrit ci-dessous dans les contraintes de forme et de validité. Pour plus d'informations, voir « 4. Structure physique ».

The version number "1.0" should be used to indicate conformance to this version of this specification; it is an error for a document to use the value "1.0" if it does not conform to this version of this specification. It is the intent of the XML working group to give later versions of this specification numbers other than "1.0", but this intent does not indicate a commitment to produce any future versions of XML, nor if any are produced, to use any particular numbering scheme. Since future versions are not ruled out, this construct is provided as a means to allow the possibility of automatic version recognition, should it become necessary. Processors may signal an error if they receive documents labeled with versions they do not support.

The function of the markup in an XML document is to describe its storage and logical structure and to associate attribute-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it.

The document type declaration must appear before the first element in the document.

Prolog

```
[22]      prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
[23]     XMLDecl ::= '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24] VersionInfo ::= S 'version' Eq (' VersionNum ' | " VersionNum ")
[25]          Eq ::= S? '=' S?
[26]  VersionNum ::= ([a-zA-Z0-9_.:]  | '-')+
[27]        Misc ::= Comment | PI |  S
```

The XML **document type declaration** contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or **DTD**. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.

A **markup declaration** is an element type declaration, an attribute-list declaration, an entity declaration, or a notation declaration. These declarations may be contained in whole or in part within parameter entities, as described in the well-formedness and validity constraints below. For fuller information, see "4. Physical Structures".

Définition de type de document

```
[28]  déclTypeDoc ::= '<!DOCTYPE' SNom (SIdExterne)? S? ('['
                      (déclBalisage | AppelEP | S)* ']' S?)? '>'
                                          [CV : Type de l'élément racine]
[29] déclBalisage ::= déclÉlément | DéclListeAtt | DéclEntité |
                      DéclNotation | IT | Commentaires
                   [CV : Imbrication stricte des déclarations et des EP ]
                                       [CF : EP dans le sous-ensemble interne ]
```

Les déclarations de balisage peuvent se composer entièrement ou partiellement du texte de remplacementd'entités paramètres. Les productions ultérieures dans cette spécification pour différents non-terminaux (`déclÉlément`, `DéclListeAtt`, et ainsi de suite) décrivent les déclarations *après* que toutes les entités paramètres aient été développées.

Contrainte de validité : Type de l'élément racine
Le `Nom` dans la déclaration de type de document doit correspondre au type d'élément de l'élément racine.

Contrainte de validité : Imbrication stricte des déclarations et des EP
Le texte de remplacement des entités paramètres doit être correctement imbriqué dans les déclarations de balisage. Si le premier ou le dernier caractère d'une déclaration de balisage (`déclBalisage` ci-dessus) est contenu dans le texte de remplacement d'un appel d'entité paramètre, tous les deux doivent être contenus dans le même texte de remplacement.

Contrainte de forme : EP dans le sous-ensemble interne
Dans le sous-ensemble interne de la DTD, les appels d'entité paramètre peuvent se produire seulement là où les déclarations de balisage sont permises, pas à l'intérieur des déclarations de balisage. (Ceci ne s'applique pas aux appels qui se produisent dans les entités paramètres externes ou au sous-ensemble externe.)

Tout comme le sous-ensemble interne, le sous-ensemble externe et les entités paramètres externes mentionnés dans la DTD doivent se composer d'une série de déclarations de balisage complètes des types permis par le symbole non-terminal `déclBalisage`, parsemées d'espaces ou d'appels d'entité paramètre. Cependant, des parties du contenu du sous-ensemble externe ou des entités paramètres externes peuvent conditionnellement être ignorées en utilisant le mécanisme de section conditionnelle ; ceci n'est pas permis dans le sous-ensemble interne.

Sous-ensemble externe

```
[30]      sousEnsembleExt ::= DéclTexte?  déclSousEnsembleExt
[31] déclSousEnsembleExt ::= ( déclBalisage |  sectConditionnelle |
                               AppelEP | S )*
```

Le sous-ensemble externe et les entités paramètres externes diffèrent également du sous-ensemble interne, les appels d'entité paramètre y étant autorisés *à l'intérieur* des déclarations de balisage, et non seulement *entre* les déclarations de balisage.

Document Type Definition

```
[28] doctypedecl ::= '<!DOCTYPE' SName (SExternalID)? S? ('['
                     (markupdecl | PEReference | S)* ']' S?)? '>'
                                                  [VC: Root Element Type]
[29]  markupdecl ::= elementdecl | AttlistDecl | EntityDecl |
                     NotationDecl | PI | Comment
                                        [VC: Proper Declaration/PE Nesting]
                                             [WFC: PEs in Internal Subset ]
```

The markup declarations may be made up in whole or in part of the replacement text of parameter entities. The productions later in this specification for individual nonterminals (`elementdecl`, `AttlistDecl`, and so on) describe the declarations *after* all the parameter entities have been included.

Validity Constraint: Root Element Type
The `Name` in the document type declaration must match the element type of the root element.

Validity Constraint: Proper Declaration/PE Nesting
Parameter-entity replacement text must be properly nested with markup declarations. That is to say, if either the first character or the last character of a markup declaration (`markupdecl` above) is contained in the replacement text for a parameter-entity reference, both must be contained in the same replacement text.

Well-Formedness Constraint: PEs in Internal Subset
In the internal DTD subset, parameter-entity references can occur only where markup declarations can occur, not within markup declarations. (This does not apply to references that occur in external parameter entities or to the external subset.)

Like the internal subset, the external subset and any external parameter entities referred to in the DTD must consist of a series of complete markup declarations of the types allowed by the non-terminal symbol `markupdecl`, interspersed with white space or parameter-entity references. However, portions of the contents of the external subset or of external parameter entities may conditionally be ignored by using the conditional section construct; this is not allowed in the internal subset.

External Subset

```
[30]      extSubset ::= TextDecl?  extSubsetDecl
[31] extSubsetDecl ::= ( markupdecl | conditionalSect |
                         PEReference | S)*
```

The external subset and external parameter entities also differ from the internal subset in that in them, parameter-entity references are permitted *within* markup declarations, not only *between* markup declarations.

Exemple de document XML avec une déclaration de type de document :

```
<?xml version="1.0"?>
<!DOCTYPE accueil SYSTEM "bonjour.dtd">
<accueil>Bonjour!</accueil>
```

L'identificateur de système « `bonjour.dtd` » donne l'URI d'une DTD pour le document.

Les déclarations peuvent également être données localement, comme dans l'exemple suivant :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE accueil [
  <!ELEMENT accueil (#PCDATA)>
]>
<accueil>Bonjour!</accueil>
```

Si les sous-ensembles externes et internes sont utilisés, le sous-ensemble interne est considéré comme se produisant avant le sous-ensemble externe. Ceci a pour effet que les déclarations d'entités et de liste d'attributs du sous-ensemble interne ont priorité sur celles du sous-ensemble externe.

2.9 Déclaration de document autonome

Les déclarations de balisage peuvent affecter le contenu du document, tel qu'il est transmis d'un processeur XML à une application ; des exemples sont des valeurs d'attribut implicites et des déclarations d'entité. La déclaration de document autonome, qui peut apparaître comme composante de la déclaration de XML, précise s'il y a de telles déclarations externes à l'entité document.

Déclaration de document autonome

```
[32] DéclDocAuto ::= S 'standalone' Égal (("'" ('yes' | 'no') "'") |
                       ('"' ('yes' | 'no') '"'))
                                        [CV : déclaration de document autonome]
```

Dans une déclaration de document autonome, la valeur « `yes` » indique qu'il n'y a pas de déclarations de balisage externes à l'entité document (dans le sous-ensemble externe de DTD, ou dans une entité paramètre externe appelée dans le sous-ensemble interne) qui affecterait l'information transmise du processeur XML à l'application. La valeur « `no` » indique qu'il y a ou peut y avoir de telles déclarations de balisage externes. Notez que la déclaration de document autonome précise seulement la présence de *déclarations* externes ; la présence, dans un document, d'appels à des *entités* externes ne change pas son caractère d'autonomie, quand ces entités sont déclarées dans le sous-ensemble interne.

S'il n'y a aucune déclaration de balisage externe, la déclaration de document autonome n'a aucune signification. S'il y a des déclarations de balisage externes mais pas de déclaration de document autonome, la valeur « `no` » est présumée.

An example of an XML document with a document type declaration:

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier "`hello.dtd`" gives the URI of a DTD for the document.

The declarations can also be given locally, as in this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

If both the external and internal subsets are used, the internal subset is considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

2.9 Standalone Document Declaration

Markup declarations can affect the content of the document, as passed from an XML processor to an application; examples are attribute defaults and entity declarations. The standalone document declaration, which may appear as a component of the XML declaration, signals whether or not there are such declarations which appear external to the document entity.

Standalone Document Declaration

```
[32] SDDecl ::= S 'standalone' Eq (("'" ('yes' | 'no') "'") |
                ('"' ('yes' | 'no') '"'))
                                        [VC: Standalone Document Declaration]
```

In a standalone document declaration, the value "`yes`" indicates that there are no markup declarations external to the document entity (either in the DTD external subset, or in an external parameter entity referenced from the internal subset) which affect the information passed from the XML processor to the application. The value "`no`" indicates that there are or may be such external markup declarations. Note that the standalone document declaration only denotes the presence of external *declarations*; the presence, in a document, of references to external *entities*, when those entities are internally declared, does not change its standalone status.

If there are no external markup declarations, the standalone document declaration has no meaning. If there are external markup declarations but there is no standalone document declaration, the value "`no`" is assumed.

Tout document XML non-autonome (`standalone="no"`) peut être converti algorithmiquement en document autonome, ce qui peut être souhaitable pour des applications diffusées en réseau.

Contrainte de validité : déclaration de document autonome
La déclaration de document autonome doit avoir la valeur « `no` » si des déclarations de balisage externes contiennent les déclarations suivantes :

— d'attributs avec des valeurs implicites, si les élément auxquels s'appliquent ces attributs apparaissent dans le document sans spécification de valeur pour ces attributs, ou
— d'entités (autres que `amp`, `lt`, `gt`, `apos` et `quot`), si des appels à ces entités apparaissent dans le document, ou
— des attributs avec des valeurs sujettes à normalisation, où l'attribut apparaît dans le document avec une valeur qui va changer en raison de la normalisation, ou
— des types d'élément avec contenu élémentaire pur, si du blanc apparaît directement dans toute instance de ce type.

Exemple de déclaration XML avec une déclaration de document autonome :

```
<?xml version="1.0" standalone='yes'?>
```

2.10 Traitement du blanc

En éditant des documents XML, il est souvent commode d'employer « du blanc » (des espaces, tabulations et interlignes, dénotés par le non-terminal `S` dans cette spécification) pour distinguer le balisage pour une plus grande lisibilité. Du tel blanc n'est pas typiquement destiné à être inclus dans la version livrée du document. D'autre part, le blanc « significatif » qui devrait être préservé dans la version livrée est courant, par exemple en poésie et en code source.

Un processeur XML doit toujours transmettre à l'application tous les caractères d'un document qui ne sont pas du balisage. Un processeur XML validateur doit également informer l'application desquels de ces caractères constituent le blanc apparaissant dans du contenu élémentaire pur.

Un attribut spécial nommé `xml:space` peut être associé à un élément pour signaler l'intention que dans cet élément, le blanc soit préservé par les applications. Dans les documents valides, cet attribut, comme tout autre, doit être déclaré s'il est utilisé. Si déclaré, il doit être donné comme type énuméré dont les seules valeurs possibles sont « `default` » et « `preserve` ». Par exemple :

```
<!ATTLIST poème   xml:space (default|preserve) 'preserve'>
```

La valeur « `default` » indique que les modes implicites de traitement du blanc sont acceptables pour cet élément ; la valeur « `preserve` » demande que les applications préservent tout le blanc. Cette intention déclarée s'applique à tous les éléments à l'intérieur du contenu de l'élément porteur de la déclaration, à moins qu'elle ne soit annulée par une autre apparition de l'attribut de `xml:space`.

L'élément racine de n'importe quel document est considéré comme n'avoir indiqué aucune intention en ce qui concerne le traitement du blanc, à moins qu'il ne fournisse une valeur pour cet attribut ou que l'attribut ne soit déclaré avec une valeur implicite.

Any XML document for which `standalone="no"` holds can be converted algorithmically to a standalone document, which may be desirable for some network delivery applications.

Validity Constraint: Standalone Document Declaration
The standalone document declaration must have the value "`no`" if any external markup declarations contain declarations of:

- attributes with default values, if elements to which these attributes apply appear in the document without specifications of values for these attributes, or
- entities (other than `amp`, `lt`, `gt`, `apos`, `quot`), if references to those entities appear in the document, or
- attributes with values subject to normalization, where the attribute appears in the document with a value which will change as a result of normalization, or
- element types with element content, if white space occurs directly within any instance of those types.

An example XML declaration with a standalone document declaration:

```
<?xml version="1.0" standalone='yes'?>
```

2.10 White Space Handling

In editing XML documents, it is often convenient to use "white space" (spaces, tabs, and blank lines, denoted by the nonterminal `S` in this specification) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space that should be preserved in the delivered version is common, for example in poetry and source code.

An XML processor must always pass all characters in a document that are not markup through to the application. A validating XML processor must also inform the application which of these characters constitute white space appearing in element content.

A special attribute named `xml:space` may be attached to an element to signal an intention that in that element, white space should be preserved by applications. In valid documents, this attribute, like any other, must be declared if it is used. When declared, it must be given as an enumerated type whose only possible values are "`default`" and "`preserve`". For example:

```
<!ATTLIST poem   xml:space (default|preserve) 'preserve'>
```

The value "`default`" signals that applications' default white-space processing modes are acceptable for this element; the value "`preserve`" indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overriden with another instance of the `xml:space` attribute.

The root element of any document is considered to have signaled no intentions as regards application space handling, unless it provides a value for this attribute or the attribute is declared with a default value.

2.11 Traitement des fins de ligne

Des entités XML analysables sont souvent enregistrées dans des fichiers qui, pour la commodité d'édition, sont organisés en lignes. Ces lignes sont typiquement séparées par une combinaison du caractère retour chariot (#xD) et retour à la ligne (#xA).

Pour simplifier la tâche des applications, quand une entité externe analysable ou une valeur littérale d'entité d'une entité analysable interne contient soit la séquence de deux caractères littéraux « #xD#xA » soit un littéral #xD, un processeur XML doit transmettre à l'application le seul caractère #xA. (Ce comportement peut simplement être produit en normalisant toutes les bris de ligne à #xA à l'entrée, avant l'analyse.)

2.12 Identification de langue

Dans le traitement de document, il est souvent utile d'identifier la langue naturelle ou formelle dans laquelle le contenu est écrit. Un attribut nommé `xml:lang` peut être inséré dans les documents pour indiquer la langue utilisée dans le contenu et dans les valeurs d'attributs de tout élément d'un document XML. Dans les documents valides, cet attribut, comme tout autre, doit être declaré s'il est utilisé. Les valeurs de l'attribut sont des identificateurs de langue tels que définis par [IETF RFC 1766], « Balises pour l'identification des langues » :

Identification de Langue

```
[33]      IdLang ::= CodeLang ('-' SousCode)*
[34]    CodeLang ::= CodeISO639 |  CodeIana |  CodeUtil
[35]  CodeISO639 ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36]    CodeIana ::= ('i' | 'I') '-' ([a-z] | [A-Z])+
[37]    CodeUtil ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38]    SousCode ::= ([a-z] | [A-Z])+
```

Le `CodeLang` peut être :

- un code de langue à deux lettres défini par [ISO 639], « Codes pour la représentation des noms des langues » ;
- un code de langue inscrit à l'Internet Assigned Numbers Authority [IANA] ; ceux-ci commencent par le préfixe « `i-` » (ou « `I-` ») ;
- un code de langue choisi par l'utilisateur ou ayant fait l'objet d'un accord entre toutes les parties pour une utilisation privée ; ceux-ci doivent commencer par le préfixe « `x-` » ou « `X-` » afin de s'assurer qu'ils ne sont pas en conflit avec des noms normalisés ultérieurement ou inscrits à l'IANA.

Il peut y avoir n'importe quel nombre de segments `SousCode` ; le premier tel sous-code, s'il existe et est formé de deux lettres, doit être un code de pays de [ISO 3166], « Codes pour la représentation des noms des pays. » Si le premier sous-code se compose de plus de deux lettres, ce doit être un code inscrit à l'IANA pour la langue en question, à moins que le `CodeLang` ne commence par le préfixe « `x-` » ou « `X-` ».

2.11 End-of-Line Handling

XML parsed entities are often stored in computer files which, for editing convenience, are organized into lines. These lines are typically separated by some combination of the characters carriage-return (#xD) and line-feed (#xA).

To simplify the tasks of applications, wherever an external parsed entity or the literal entity value of an internal parsed entity contains either the literal two-character sequence "#xD#xA" or a standalone literal #xD, an XML processor must pass to the application the single character #xA. (This behavior can conveniently be produced by normalizing all line breaks to #xA on input, before parsing.)

2.12 Language Identification

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named `xml:lang` may be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document. In valid documents, this attribute, like any other, must be declared if it is used. The values of the attribute are language identifiers as defined by [IETF RFC 1766], "Tags for the Identification of Languages":

Language Identification

```
[33] LanguageID ::= Langcode ('-' Subcode)*
[34]   Langcode ::= ISO639Code |  IanaCode |  UserCode
[35] ISO639Code ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36]   IanaCode ::= ('i' | 'I') '-' ([a-z] | [A-Z])+
[37]   UserCode ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38]    Subcode ::= ([a-z] | [A-Z])+
```

The `Langcode` may be any of the following:

- a two-letter language code as defined by [ISO 639], "Codes for the representation of names of languages"
- a language identifier registered with the Internet Assigned Numbers Authority [IANA]; these begin with the prefix "`i-`" (or "`I-`")
- a language identifier assigned by the user, or agreed on between parties in private use; these must begin with the prefix "`x-`" or "`X-`" in order to ensure that they do not conflict with names later standardized or registered with IANA

There may be any number of `Subcode` segments; if the first subcode segment exists and the Subcode consists of two letters, then it must be a country code from [ISO 3166], "Codes for the representation of names of countries." If the first subcode consists of more than two letters, it must be a subcode for the language in question registered with IANA, unless the `Langcode` begins with the prefix "`x-`" or "`X-`".

Il est usuel de donner le code de langue en minuscules, et le code de pays (s'il y a lieu) en majuscules. Notez que ces valeurs, à la différence d'autres noms dans les documents XML, ne sont pas sensibles à la casse.

Exemples :

```
<p xml:lang="fr">Cachez ce sein que je ne saurais voir.</p>
<p xml:lang="fr-FR">Ce programme a une bogue.</p>
<p xml:lang="fr-CA">Ce programme a un bogue.</p>
<sp qui="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit heiflem Bem,h'n.</l>
  </sp>
```

L'intention déclarée avec `xml:lang` est considérée s'appliquer à tous les attributs et au contenu de l'élément où on l'indique, à moins qu'elle ne soit annulée par une apparition de `xml:lang` sur un autre élément dans ce contenu.

Une déclaration simple pour `xml:lang` pourrait prendre la forme

```
xml:lang  NMTOKEN  #IMPLIED
```

mais des valeurs implicites spécifiques peuvent également être attribuées, si appropriées. Dans une collection de poésie française pour étudiants anglais, avec des gloses et des notes en anglais, l'attribut `xml:lang` pourrait être déclaré de cette façon :

```
<!ATTLIST poésie xml:lang NMTOKEN 'fr'>
<!ATTLIST glose  xml:lang NMTOKEN 'en'>
<!ATTLIST note   xml:lang NMTOKEN 'en'>
```

3. Structures logiques

Chaque document XML contient un ou plusieurs **éléments**, dont les limites sont marquées soit par des balises ouvrantes et fermantes, soit, pour les éléments vides, par une balise d'élément vide. Chaque élément a un type, identifié par un nom, parfois appelé son « identificateur générique » (IG), on peut y associer un jeu de spécifications d'attribut. Chaque spécification d'attribut comprend un nom et une valeur.

Élément

```
[39] élément ::= BaliseÉlémVide
                | BaliseO contenu BaliseF
                                  [CF :Correspondance de type d'élément]
                                                    [CV :Élément valide]
```

It is customary to give the language code in lower case, and the country code (if any) in upper case. Note that these values, unlike other names in XML documents, are case insensitive.

For example:

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit hei$em Bemüh'n.</l>
</sp>
```

The intent declared with `xml:lang` is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of `xml:lang` on another element within that content.

A simple declaration for `xml:lang` might take the form

```
xml:lang  NMTOKEN  #IMPLIED
```

but specific default values may also be given, if appropriate. In a collection of French poems for English students, with glosses and notes in English, the xml:lang attribute might be declared this way:

```
<!ATTLIST poem   xml:lang NMTOKEN 'fr'>
<!ATTLIST gloss  xml:lang NMTOKEN 'en'>
<!ATTLIST note   xml:lang NMTOKEN 'en'>
```

3. Logical Structures

Each XML document contains one or more **elements**, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and may have a set of attribute specifications. Each attribute specification has a name and a value.

Element

```
[39] element ::= EmptyElemTag
                 | STag content ETag
                                                [WFC: Element Type Match]
                                                      [VC: Element Valid]
```

Cette spécification ne contraint pas la sémantique, l'utilisation ou (au-delà de la syntaxe) les noms des types d'éléments ou d'attributs, hormis le fait que les noms qui commencent par `(('X'|'x')('M'|'m')('L'|'l'))` sont réservés à des fins de standardisation dans cette version ou d'ultérieures versions de cette spécification.

Contrainte de forme : Correspondance de type d'élément
Le `Nom` dans une balise fermante d'un élément doit correspondre au type d'élément de la balise ouvrante.

Contrainte de validité : Élément valide
Un élément est valide s'il existe une déclaration correspondant à `déclÉlément` où le `nom` correspond à un type d'élément, et l'une des conditions suivantes est vraie :

1. La déclaration correspond à `EMPTY` et l'élément n'a pas de contenu.
2. La déclaration correspond à `sousÉléments` et la suite de sous-éléments appartient au langage engendré par l'expression régulière du modèle de contenu, avec du blanc optionnel (des caractères correspondant au non-terminal `S`) entre chaque paire de sous-éléments.
3. La déclaration correspond à `Mixte` et le contenu est constitué de données textuelles et de sous-éléments dont les types correspondent aux noms dans le modèle du contenu.
4. La déclaration correspond à `ANY` et le type de chaque sous-élément a été déclaré.

3.1 Balises ouvrantes, balises fermantes et balises d'élément vide

Le début de chaque élément XML non vide est marqué d'une **balise ouvrante (ou balise de début)**.

Balise ouvrante

```
[40]  BaliseO ::= '<'Nom(S Attribut)* S? '>'
                                        [CF :Spécif.  unique de l'attribut]
[41] Attribut ::= Nom Égal ValeurAtt
                                            [CV :Type valeur de l'attribut]
                                         [CF :Pas d'appel d'entité externe]
                                  [CF :Pas de < dans les valeurs d'attribut]
```

Le `Nom` des balises de début et de fin spécifie le **type** de l'élément. Les couples `Nom-ValeurAtt` constituent les **spécifications d'attribut** de l'élément, avec pour chaque couple le `Nom` désigné par le **nom de l'attribut**, et le contenu de `ValeurAtt` (le texte compris entre les délimiteurs « ' » ou « " ») désigné par la **valeur de l'attribut**.

Contrainte de forme : Spécification unique de l'attribut
Aucun nom d'attribut ne peut apparaître plus d'une fois dans la même balise de début ou d'élément vide.

Contrainte de validité : Type de valeur de l'attribut
L'attribut doit avoir été déclaré; la valeur doit correspondre au type déclaré pour cet attribut. (Pour les types d'attribut, voir « 3.3 Déclarations des listes d'attributs ».)

This specification does not constrain the semantics, use, or (beyond syntax) names of the element types and attributes, except that names beginning with a match to `(('X'|'x')('M'|'m')('L'|'l'))` are reserved for standardization in this or future versions of this specification.

Well-Formedness Constraint: Element Type Match
The `Name` in an element's end-tag must match the element type in the start-tag.

Validity Constraint: Element Valid
An element is valid if there is a declaration matching `elementdecl` where the `Name` matches the element type, and one of the following holds:

1. The declaration matches `EMPTY` and the element has no content.
2. The declaration matches `children` and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal `S`) between each pair of child elements.
3. The declaration matches `Mixed` and the content consists of character data and child elements whose types match names in the content model.
4. The declaration matches `ANY`, and the types of any child elements have been declared.

3.1 Start-Tags, End-Tags, and Empty-Element Tags

The beginning of every non-empty XML element is marked by a **start-tag**.

Start-tag

```
[40]       STag ::= '<' Name (S Attribute)* S? '>'
                                                    [WFC: Unique Att Spec]
[41] Attribute ::= Name Eq AttValue
                                                [VC: Attribute Value Type]
                                        [WFC: No External Entity References]
                                             [WFC: No < in Attribute Values]
```

The `Name` in the start- and end-tags gives the element's **type**. The `Name-AttValue` pairs are referred to as the **attribute specifications** of the element, with the `Name` in each pair referred to as the **attribute name** and the content of the `AttValue` (the text between the `'` or `"` delimiters) as the **attribute value**.

Well-Formedness Constraint: Unique Att Spec
No attribute name may appear more than once in the same start-tag or empty-element tag.

Validity Constraint: Attribute Value Type
The attribute must have been declared; the value must be of the type declared for it. (For attribute types, see "3.3 Attribute-List Declarations".)

Contrainte de forme : Pas d'appel d'entité externe
Les valeurs d'attribut ne peuvent contenir d'appels d'entité directs ou indirects à des entités externes.

Contrainte de forme : Pas de<dans les valeurs d'attribut
Le texte de remplacement de toute entité appelée directement ou indirectement dans une valeur d'attribut (autre que « `<` ») ne peut contenir un `<`.

Exemple de balise ouvrante:

```
<termdef id="dt-chien" terme="chien">
```

La fin de chaque élément qui commence par une balise de début doit être marqué d'une **balise fermante (ou balise de fin)** contenant un nom qui renvoie au type de l'élément spécifié dans la balise de début :

Balise fermante

```
[42] BaliseF ::= '</' Nom S? '>'
```

Exemple de balise fermante :

```
</termdef>
```

On appelle le texte compris entre les balises de début et de fin le **contenu** de l'élément :

Contenu des éléments

```
[43] contenu ::= ( élément | DonnéesTextuelles | Appel | SectionDT |
                 IT | Commentaire )*
```

Si un élément est **vide**, il doit être indiqué soit par une balise ouvrante suivie immédiatement d'une balise fermante, soit par une balise d'élément vide. Une **balise d'élément vide** se formule d'une manière particulière :

Balises pour éléments vides

```
[44] BaliseÉlemVide ::= '<' Nom (S Attribut)* S? '/>'
                                        [CF :Spécif.  unique de l'attribut]
```

Les balises d'élement vide peuvent être utilisées pour tout élément qui n'a pas de contenu, qu'il ait été déclaré ou non avec le mot-clé `EMPTY`. À des fins d'interopérabilité, la balise d'élément vide doit être utilisée, et ne peut être utilisée que, pour les éléments qui ont été déclarés`EMPTY`.

Exemples d'éléments vides :

```
<IMG align="left"
 src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

Well-Formedness Constraint: No External Entity References
Attribute values cannot contain direct or indirect entity references to external entities.

Well-Formedness Constraint: No < in Attribute Values
The replacement text of any entity referred to directly or indirectly in an attribute value (other than "`<`") must not contain a <.

An example of a start-tag:

```
<termdef id="dt-dog" term="dog">
```

The end of every element that begins with a start-tag must be marked by an **end-tag** containing a name that echoes the element's type as given in the start-tag:

End-tag

```
[42] ETag ::= '</' Name S? '>'
```

An example of an end-tag:

```
</termdef>
```

The text between the start-tag and end-tag is called the element's **content**:

Content of Elements

```
[43] content ::= (element | CharData | Reference | CDSect |
                  PI | Comment)*
```

If an element is **empty**, it must be represented either by a start-tag immediately followed by an end-tag or by an empty-element tag. An **empty-element tag** takes a special form:

Tags for Empty Elements

```
[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'
                                                [WFC: Unique Att Spec]
```

Empty-element tags may be used for any element which has no content, whether or not it is declared using the keyword `EMPTY`. For interoperability, the empty-element tag must be used, and can only be used, for elements which are declared`EMPTY`.

Examples of empty elements:

```
<IMG align="left"
 src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

3.2 Déclarations de type d'élément

La structure des éléments d'un document XML peut, à des fins de validation, être contrainte à l'aide de déclarations de type d'élément et de liste d'attributs. La déclaration de type d'un élément contraint le contenu de cet élément.

Les déclarations de type d'un élément limitent habituellement les types d'élément qui peuvent apparaître comme sous-éléments de celui-ci. Au choix de l'utilisateur, un processeur XML peut émettre un avertissement quand une déclaration mentionne un type d'élément pour lequel aucune déclaration n'a été fournie, mais ceci ne constitue pas une erreur.

Une **déclaration de type d'élément** se formule de la façon suivante :

Déclaration de type d'élément

```
[45] déclÉlément ::= '<!ELEMENT' S Nom S specContenu S? '>'
                                  [CV :Déclaration de type d'élément unique]
[46] specContenu ::= 'EMPTY' | 'ANY' | Mixte| sousÉléments
```

où le `Nom` fournit le type d'élément que l'on déclare.

Contrainte de validité : Déclaration de type d'élément unique
Aucun type d'élément ne peut être déclaré plus d'une fois.

Exemples de déclarations de type d'élément :

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %param.nom; %param.contenu; >
<!ELEMENT contenant ANY>
```

3.2.1 Contenu élémentaire pur Un type d'élément a un **contenu élémentaire pur** quand des éléments de ce type ne peuvent contenir que des sous-éléments, (pas de données textuelles), séparés par du blanc (caractères correspondant au non-terminal `S`) facultatif. Dans ce cas, la contrainte comprend un modèle de contenu, une grammaire simple régissant les types admis pour les sous-éléments et l'ordre dans lequel ceux-ci peuvent apparaître. Cette grammaire est construite à l'aide de particules de contenu (`pc`), constituées de noms, de listes de choix de particules de contenu, ou de listes de suites de particules de contenu :

Modèles de contenu élémentaire pur

```
[47] sousÉléments ::= ( choix |  séq ) ('?'  | '*' | '+')?
[48]           pc ::= ( Nom |  choix |  séq ) ('?'  | '*' | '+')?
[49]        choix ::= '(' S? pc ( S? '|' S? pc )* S? ')'
                          [CV :Imbrication stricte des parenthèses dans EP]
[50]          séq ::= '(' S? pc ( S? ',' S? pc )* S? ')'
                          [CV :Imbrication stricte des parenthèses dans EP]
```

3.2 Element Type Declarations

The element structure of an XML document may, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.

Element type declarations often constrain which element types can appear as children of the element. At user option, an XML processor may issue a warning when a declaration mentions an element type for which no declaration is provided, but this is not an error.

An **element type declaration** takes the form:

Element Type Declaration

```
[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>'
                                        [VC: Unique Element Type Declaration]
[46] contentspec ::= 'EMPTY' | 'ANY' |  Mixed | children
```

where the `Name` gives the element type being declared.

Validity Constraint: Unique Element Type Declaration
No element type may be declared more than once.

Examples of element type declarations:

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

3.2.1 Element Content An element type has **element content** when elements of that type must contain only child elements (no character data), optionally separated by white space (characters matching the nonterminal `S`). In this case, the constraint includes a content model, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear. The grammar is built on content particles (cps), which consist of names, choice lists of content particles, or sequence lists of content particles:

Element-content Models

```
[47] children ::= (choice | seq) ('?'  | '*' | '+')?
[48]       cp ::= (Name | choice | seq) ('?'  | '*' | '+')?
[49]   choice ::= '(' S? cp ( S? '|' S? cp )* S? ')'
                                              [VC: Proper Group/PE Nesting]
[50]      seq ::= '(' S? cp ( S? ',' S? cp )* S? ')'
                                              [VC: Proper Group/PE Nesting]
```

où chaque `Nom` est le type d'un élément qui peut apparaître comme sous-élément. Une particule de contenu dans une liste de choix peut apparaître au sein d'un contenu élémentaire pur à l'endroit où une liste de choix apparaît dans la grammaire; les particules de contenu présentes dans une liste de suite doivent toutes apparaître dans le contenu élémentaire pur dans l'ordre précisé dans la liste. Les caractères optionnels qui suivent un nom ou une liste déterminent si l'élément ou les particules de contenu dans la liste peuvent apparaître une fois ou plus (`+`), zéro fois ou plus (`*`), ou bien au maximum une fois(`?`). L'absence d'un tel opérateur signifie que l'élément ou la particule de contenu doit apparaître exactement une fois. Leur syntaxe et leur sens sont identiques à ceux qui sont utilisés dans les productions de cette spécification.

Le contenu d'un élément correspond à un modèle de contenu si et seulement si l'on peut tracer un chemin à travers le modèle de contenu qui respecte les opérateurs de suite, de choix et de répétition et qui fait correspondre chaque élément du contenu à un type d'élément défini dans le modèle de contenu. À des fins de compatibilité, le fait qu'un élément dans le document puisse correspondre à plus d'une occurrence de ce type d'élément dans le modèle de contenu constitue une erreur. Pour de plus amples informations, voir « E. Modèles de contenu déterministes ».

Contrainte de validité : Imbrication stricte des parenthèses dans EP
Les parenthèses des textes de remplacement d'une entité paramètre doivent être strictement imbriqués. Ceci signifie que si la parenthèse ouvrante ou fermante d'une production `choix`, `séq` ou `Mixte` se retrouve dans le texte de remplacement d'une entité paramètre, ces deux parenthèses doivent être contenues dans le même texte de remplacement. À des fins d'interopérabilité, si un appel d'entité paramètre apparaît dans une production `choix`, `séq` ou `Mixte`, son texte de remplacement ne peut pas être vide, le dernier caractère significatif (non blanc) du texte de remplacement ne peut pas non plus être un connecteur (`|` ou `,`).

Exemples de modèles de contenu élémentaire pur :

```
<!ELEMENT stipu (préface, corps, postface?)>
<!ELEMENT div1 (entête, (p | liste | note)*, div2*)>
<!ELEMENT corps-dictionnaire (%div.mélange; | %dict.mélange;)*>
```

3.2.2 Contenu mixte

Un type d'élément a un **contenu mixte** quand des éléments de ce type peuvent contenir des données textuelles, parsemées, le cas échéant, de sous-éléments. Dans ce cas, les types des sous-éléments peuvent être contraints mais pas leur ordre ni leur nombre.

Déclaration de contenu mixte

```
[51] Mixte ::= '(' S ? '#PCDATA' ( S? '|' S? Nom )* S? ')*' |
               '(' S? '#PCDATA' S? ')'
                       [CV : Imbrication stricte des parenthèses dans EP]
                                                    [CV : Type unique]
```

où les `Nom`s fournissent les types des éléments qui peuvent apparaître comme sous-éléments.

Contrainte de validité : Type unique
Le même nom ne peut apparaître plus d'une fois dans une seule déclaration de contenu mixte.

where each `Name` is the type of an element which may appear as a child. Any content particle in a choice list may appear in the element content at the location where the choice list appears in the grammar; content particles occurring in a sequence list must each appear in the element content in the order given in the list. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more (`+`), zero or more (`*`), or zero or one times (`?`). The absence of such an operator means that the element or content particle must appear exactly once. This syntax and meaning are identical to those used in the productions in this specification.

The content of an element matches a content model if and only if it is possible to trace out a path through the content model, obeying the sequence, choice, and repetition operators and matching each element in the content against an element type in the content model. For compatibility, it is an error if an element in the document can match more than one occurrence of an element type in the content model. For more information, see "E. Deterministic Content Models".

Validity Constraint: Proper Group/PE Nesting
Parameter-entity replacement text must be properly nested with parenthetized groups. That is to say, if either of the opening or closing parentheses in a `choice`, `seq`, or `Mixed` construct is contained in the replacement text for a parameter entity, both must be contained in the same replacement text. For interoperability, if a parameter-entity reference appears in a `choice`, `seq`, or `Mixed` construct, its replacement text should not be empty, and neither the first nor last non-blank character of the replacement text should be a connector (`|` or `,`).

Examples of element-content models:

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2 Mixed Content

An element type has **mixed content** when elements of that type may contain character data, optionally interspersed with child elements. In this case, the types of the child elements may be constrained, but not their order or their number of occurrences:

Mixed-content Declaration

```
[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' |
               '(' S? '#PCDATA' S? ')'
                                              [VC: Proper Group/PE Nesting]
                                                   [VC: No Duplicate Types]
```

where the `Name`s give the types of elements that may appear as children.

Validity Constraint: No Duplicate Types
The same name must not appear more than once in a single mixed-content declaration.

Exemples de déclarations de contenu mixte :

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %police; | %proposition; |
                       %spécial; | %formulaire;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 Déclarations de liste d'attributs

On utilise des attributs pour associer des couples nom-valeur aux éléments. Les spécifications d'attribut ne peuvent apparaître qu'au sein de balises ouvrantes et de balises d'élément vide ; ainsi, les productions utilisées pour les reconnaître apparaissent dans « 3.1 Balises ouvrantes, balises fermantes et balises d'élément vide ». Les déclarations de liste d'attributs peuvent servir à :

— définir un jeu d'attributs se rapportant à un type d'élément donné;
— établir des contraintes de type pour ces attributs;
— fournir des valeurs implicites pour des attributs.

Les déclarations de liste d'attributs précisent le nom, le type de données et la valeur implicite (le cas échéant) de chaque attribut associé à un type d'élément donné :

Déclaration de liste d'attributs

```
[52] DéclListeAtt ::= '<!ATTLIST' S Nom DéfAtt* S? '>'
[53]       DéfAtt ::= S Nom S TypeAtt S DéclValImpl
```

Le `Nom` dans la production `DéclListeAtt` correspond au type d'un élément. Au choix de l'utilisateur, un processeur XML peut émettre un avertissement si on déclare des attributs pour un type d'élément qui lui-même n'est pas déclaré, mais ceci ne constitue pas une erreur. Le `Nom` que l'on retrouve dans la règle `DéfAtt` correspond au nom de l'attribut.

Quand plus d'une `DéclListeAtt` existe pour un type d'élément donné, le contenu de toutes les déclarations fournies est fusionnée. Quand plus d'une définition existe pour un même attribut d'un type d'élément donné, seule la première déclaration compte, les déclarations subséquentes sont ignorées. À des fins d'interopérabilité, les rédacteurs des DTDs pourraient décider de fournir au plus une déclaration de liste d'attributs pour un type d'élément donné, au plus une définition d'attribut pour un nom d'attribut donné et au moins une définition d'attribut pour chaque déclaration de liste d'attributs. De même, à des fins d'interopérabilité, un processeur XML pourra au gré de l'utilisateur émettre un avertissement quand il existe plus d'une déclaration de liste d'attributs pour un type d'élément donné ou quand plus d'une définition d'attribut existe pour un attribut donné, mais ceci ne constitue pas une erreur.

3.3.1 Types d'attribut Les types d'attribut XML sont de trois genres : un type chaîne, une série de types atomiques et des types énumérés. Le type chaîne peut prendre comme valeur toute chaîne littérale ; les types atomiques possèdent différentes contraintes lexicales et sémantiques précisées ci-dessous :

Examples of mixed content declarations:

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 Attribute-List Declarations

Attributes are used to associate name-value pairs with elements. Attribute specifications may appear only within start-tags and empty-element tags; thus, the productions used to recognize them appear in "3.1 Start-Tags, End-Tags, and Empty-Element Tags". Attribute-list declarations may be used:

- To define the set of attributes pertaining to a given element type.
- To establish type constraints for these attributes.
- To provide default values for attributes.

Attribute-list declarations specify the name, data type, and default value (if any) of each attribute associated with a given element type:

Attribute-list Declaration

```
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'
[53]      AttDef ::= S Name S AttType S DefaultDecl
```

The `Name` in the `AttlistDecl` rule is the type of an element. At user option, an XML processor may issue a warning if attributes are declared for an element type not itself declared, but this is not an error. The `Name` in the `AttDef` rule is the name of the attribute.

When more than one `AttlistDecl` is provided for a given element type, the contents of all those provided are merged. When more than one definition is provided for the same attribute of a given element type, the first declaration is binding and later declarations are ignored. For interoperability, writers of DTDs may choose to provide at most one attribute-list declaration for a given element type, at most one attribute definition for a given attribute name, and at least one attribute definition in each attribute-list declaration. For interoperability, an XML processor may at user option issue a warning when more than one attribute-list declaration is provided for a given element type, or more than one attribute definition is provided for a given attribute, but this is not an error.

3.3.1 Attribute Types XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints, as noted:

Types d'attribut

```
[54]      TypeAtt ::= TypeChaîne| TypeAtomique| TypeÉnuméré
[55]   TypeChaîne ::= 'CDATA'
[56] TypeAtomique ::= 'ID'                                        [CV : ID]
                                    [CV :Un seul ID par type d'élément]
                                 [CV :Valeur implicite de l'attribut ID]
                      | 'IDREF'                                   [CV : IDREF]
                      | 'IDREFS'                                  [CV : IDREF]
                      | 'ENTITY'                          [CV : Nom d'entité]
                      | 'ENTITIES'                        [CV : Nom d'entité]
                      | 'NMTOKEN'                        [CV : Atome nominal]
                      | 'NMTOKENS'                       [CV : Atome nominal]
```

Contrainte de validité : ID
Les valeurs de type `ID` doivent correspondre à la production `Nom`. Un nom ne peut apparaître plus d'une fois dans un document XML comme la valeur de ce type, i.e. les valeurs ID doivent identifier uniquement les éléments qu'elles désignent..

Contrainte de validité : Un seul ID par type d'élément
Aucun type d'élément ne peut posséder plus d'un attribut ID.

Contrainte de validité : Valeur implicite de l'attribut ID
Un attribut ID doit avoir comme valeur implicite déclarée soit `#IMPLIED` soit `#REQUIRED`.

Contrainte de validité : IDREF
Les valeurs de type `IDREF` doivent correspondre à la production `Nom`, et les valeurs de type `IDREFS` doivent correspondre à des `Noms` ; chaque `Nom` doit correspondre à la valeur d'un attribut ID sur un élément quelconque du document XML, en d'autres mots les valeurs `IDREF` doivent correspondre à la valeur d'un attribut ID.

Contrainte de validité : Nom d'entité
Les valeurs de type `ENTITY` doivent correspondre à la production `Nom` , les valeurs de type `ENTITIES` doivent correspondre à des `Noms` ; chaque `Nom` doit correspondre au nom d'une entitée non-analysable déclarée dans la DTD.

Contrainte de validité : Atome nomimal
Les valeurs de type `NMTOKEN` doivent correspondre à la production `AtomeNml` ; les valeurs de type `NMTOKENS` doivent correspondre à des AtomeNmx.

Les attributs énumérés peuvent prendre une valeur parmi une liste de valeurs fournie dans la déclaration. Il existe deux sortes de types énumérés :

Types d'attributs énumérés

```
[57]  TypeÉnuméré ::= TypeNotation| Énumération
[58] TypeNotation ::= 'NOTATION' S '(' S? Nom ( S? '|' S? Nom )* S? ')'
                                              [CV : Attributs de notation]
[59]  Énumération ::= '(' S? AtomeNml ( S? '|' S? AtomeNml )* S? ')'
                                                      [CV : Énumération]
```

Attribute Types

```
[54]       AttType ::= StringType | TokenizedType | EnumeratedType
[55]    StringType ::= 'CDATA'
[56] TokenizedType ::= 'ID'                                    [VC: ID]
                                          [VC: One ID per Element Type]
                                             [VC: ID Attribute Default]
                     | 'IDREF'                                 [VC: IDREF]
                     | 'IDREFS'                                [VC: IDREF]
                     | 'ENTITY'                          [VC: Entity Name]
                     | 'ENTITIES'                         [VC: Entity Name
                     | 'NMTOKEN'                          [VC: Name Token]
                     | 'NMTOKENS'                         [VC: Name Token]
```

Validity Constraint: ID
Values of type `ID` must match the `Name` production. A name must not appear more than once in an XML document as a value of this type; i.e., ID values must uniquely identify the elements which bear them.

Validity Constraint: One ID per Element Type
No element type may have more than one ID attribute specified.

Validity Constraint: ID Attribute Default
An ID attribute must have a declared default of `#IMPLIED` or `#REQUIRED`.

Validity Constraint: IDREF
Values of type `IDREF` must match the `Name` production, and values of type `IDREFS` must match `Names`; each `Name` must match the value of an ID attribute on some element in the XML document; i.e. `IDREF` values must match the value of some ID attribute.

Validity Constraint: Entity Name
Values of type `ENTITY` must match the `Name` production, values of type `ENTITIES` must match `Names`; each `Name` must match the name of an unparsed entity declared in the DTD.

Validity Constraint: Name Token
Values of type `NMTOKEN` must match the `Nmtoken` production; values of type `NMTOKENS` must match Nmtokens.

Enumerated attributes can take one of a list of values provided in the declaration. There are two kinds of enumerated types:

Enumerated Attribute Types

```
[57] EnumeratedType ::= NotationType | Enumeration
[58]   NotationType ::= 'NOTATION' S '(' S? Name (S? '|' S? Name)* S? ')'
                                                  [VC: Notation Attributes]
[59]    Enumeration ::= '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
                                                       [VC: Enumeration]
```

Un attribut `NOTATION` identifie une notation, déclarée dans la DTD conjointement avec ses identificateurs systèmes et publics, que l'on utilisera pour interpréter l'élément auquel l'attribut est joint.

Contrainte de validité : Attributs de notation
Les valeurs de ce type doivent correspondre à un des noms de notation inclus dans la déclaration ; tous les noms de notation dans la déclaration doivent être déclarés.

Contrainte de validité : Énumération
Les valeurs de ce type doivent correspondre à un des atomes `AtomeNml` dans la déclaration.

À des fins d'interopérabilité, le même `AtomeNml` ne devrait pas apparaître plus d'une fois dans les types d'attribut énuméré d'un même type d'élément.

3.3.2 Valeurs implicites des attributs Une déclaration d'attribut précise si la présence de l'attribut est exigée et, si elle ne l'est pas, précise le comportement du processeur XML quand l'attribut déclaré est absent dans un document.

Valeurs implicites des attributs

```
[60] DéclValImpl ::= '#REQUIRED' | '#IMPLIED' |
                     (('#FIXED' S)? ValeurAtt)
                                          [CV : Attribut obligatoire]
                               [CV : Valeur implicite de l'attribut permise]
                                     [CF : Pas de < dans valeurs d'attribut]
                                  [CV : Valeur implicite de l'attribut fixe]
```

Dans une déclaration d'attribut, `#REQUIRED` signifie que l'attribut doit toujours être fourni, `#IMPLIED` qu'aucune valeur implicite n'est fournie. Si la déclaration n'est ni `#REQUIRED` ni `#IMPLIED` alors la valeur de `ValeurAtt` précise la valeur **implicite** déclarée; le mot-clé `#FIXED` indique que l'attribut doit toujours avoir la valeur implicite. Si une valeur implicite est déclarée, le processeur XML doit se comporter comme si l'attribut était présent et égal à la valeur implicite déclarée lorsqu'il s'aperçoit de l'absence d'un attribut.

Contrainte de validité : Attribut obligatoire
Si la déclaration implicite est le mot-clé `#REQUIRED` il faut alors préciser l'attribut pour tous les éléments du type dans la déclaration de la liste d'attributs.

Contrainte de validité : Valeur implicite de l'attribut permise
La valeur implicite déclarée doit satisfaire aux contraintes lexicales du type d'attribut déclaré.

Contrainte de validité : Valeur implicite de l'attribut fixe
Si un attribut a une valeur implicite déclarée avec le mot-clé `#FIXED`, les instances de cet attribut doivent correspondre à la valeur implicite.

Exemples de déclarations de liste d'attributs :

```
<!ATTLIST défterme
          ident   ID      #REQUIRED
          nom     CDATA   #IMPLIED>
<!ATTLIST liste
          type    (àpuces|ordonnée|glossaire)  "ordonnée">
<!ATTLIST formulaire
          méthode CDATA   #FIXED "ENVOI">
```

A NOTATION attribute identifies a notation, declared in the DTD with associated system and/or public identifiers, to be used in interpreting the element to which the attribute is attached.

Validity Constraint: Notation Attributes
Values of this type must match one of the notation names included in the declaration; all notation names in the declaration must be declared.

Validity Constraint: Enumeration
Values of this type must match one of the Nmtoken tokens in the declaration.

For interoperability, the same Nmtoken should not occur more than once in the enumerated attribute types of a single element type.

3.3.2 Attribute Defaults An attribute declaration provides information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document.

Attribute Defaults

```
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED' |
                     (('#FIXED' S)? AttValue)    [VC: Required Attribute]
                                                   [VC: Attribute Default Legal]
                                                 [WFC: No < in Attribute Values]
                                                   [VC: Fixed Attribute Default]
```

In an attribute declaration, #REQUIRED means that the attribute must always be provided, #IMPLIED that no default value is provided. If the declaration is neither #REQUIRED nor #IMPLIED, then the AttValue value contains the declared **default** value; the #FIXED keyword states that the attribute must always have the default value. If a default value is declared, when an XML processor encounters an omitted attribute, it is to behave as though the attribute were present with the declared default value.

Validity Constraint: Required Attribute
If the default declaration is the keyword #REQUIRED, then the attribute must be specified for all elements of the type in the attribute-list declaration.

Validity Constraint: Attribute Default Legal
The declared default value must meet the lexical constraints of the declared attribute type.

Validity Constraint: Fixed Attribute Default
If an attribute has a default value declared with the #FIXED keyword, instances of that attribute must match the default value.

Examples of attribute-list declarations:

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>
<!ATTLIST list
          type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
          method  CDATA   #FIXED "POST">
```

3.3.3 Normalisation de valeur d'attribut Avant que la valeur d'un attribut ne soit passée à l'application ou que l'on vérifie sa validité, le processeur XML doit normaliser cette valeur de la façon suivante :

— on traite l'appel de caractère en ajoutant les caractères appelés à la fin de la valeur d'attribut ;
— on traite l'appel d'entité en traitant récursivement le texte de remplacement de l'entité ;
— on traite un séparateur (#x20, #xD, #xA, #x9) en ajoutant un #x20 à la la valeur normalisée; il faut cependant noter qu'on ajoutera un seul #x20 pour une suite « #xD#xA » faisant partie d'une entité analysable externe ou de la valeur littérale d'entité d'une entité analysable interne ;
— on traite les autres caractères en les ajoutant à la la valeur normalisée.

Si la valeur déclarée n'est pas CDATA alors le processeur XML devra poursuivre le traitement de la valeur normalisée de l'attribut en se défaisant des blancs (#x20) de tête et de queue et en replaçant les suites de blancs (#x20) par un seul caractère blanc (#x20).

Tous les attributs pour lesquels on a lu aucune déclaration devrait être considéré par un processeur non-validateur comme si on les avait déclarés au moyen de `CDATA`.

3.4 Sections conditionnelles

Les sections conditionnelles sont des portions du sous-ensemble externe de la déclaration de type de document qui, selon la valeur d'un mot-clé, sont incluses dans la structure logique de la DTD ou excluses de celles-ci.

Section conditionnelle

```
[61] SectConditionnelle ::= sectInclude| sectIgnore
[62]        sectInclude ::= '<![' S? 'INCLUDE' S? '['
                            déclSousEnsembleExt ']]>'
[63]         sectIgnore ::= '<![' S? 'IGNORE' S? '['
                            contenuSectIgnore* ']]>'
[64]  contenuSectIgnore ::= Ignore ('<![' contenuSectIgnore
                            ']]>'Ignore)*
[65]             Ignore ::= Car* - ( Car* ('<![' | ']]>') Car*)
```

À l'instar des sous-ensembles internes et externes de DTD, une section conditionnelle peut contenir une ou plusieurs instances complètes de déclarations, de commentaires, d'instructions de traitement ou de sections conditionnelles imbriquées, le tout parsemé de séparateurs (des blancs).

Si le mot-clé d'une section conditionnelle est `INCLUDE` alors le contenu de la section conditionnelle fait partie de la DTD. Si le mot-clé de la section conditionnelle est `IGNORE` alors le contenu de la section conditionnelle ne fait pas partie, au niveau logique, de la DTD.

3.3.3 Attribute-Value Normalization Before the value of an attribute is passed to the application or checked for validity, the XML processor must normalize it as follows:

- a character reference is processed by appending the referenced character to the attribute value
- an entity reference is processed by recursively processing the replacement text of the entity
- a whitespace character (#x20, #xD, #xA, #x9) is processed by appending #x20 to the normalized value, except that only a single #x20 is appended for a "#xD#xA" sequence that is part of an external parsed entity or the literal entity value of an internal parsed entity
- other characters are processed by appending them to the normalized value

If the declared value is not CDATA, then the XML processor must further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

All attributes for which no declaration has been read should be treated by a non-validating parser as if declared `CDATA`.

3.4 Conditional Sections

Conditional sections are portions of the document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.

Conditional Section

```
[61]    conditionalSect ::= includeSect | ignoreSect
[62]        includeSect ::= '<![' S? 'INCLUDE' S? '['
                            extSubsetDecl ']]>'
[63]         ignoreSect ::= '<![' S? 'IGNORE' S? '['
                            ignoreSectContents* ']]>'
[64] ignoreSectContents ::= Ignore ('<![' ignoreSectContents
                            ']]>' Ignore)*
[65]             Ignore ::= Char* - (Char* ('<![' | ']]>') Char*)
```

Like the internal and external DTD subsets, a conditional section may contain one or more complete declarations, comments, processing instructions, or nested conditional sections, intermingled with white space.

If the keyword of the conditional section is `INCLUDE`, then the contents of the conditional section are part of the DTD. If the keyword of the conditional section is `IGNORE`, then the contents of the conditional section are not logically part of the DTD.

Remarquons qu'afin de rendre l'analyse robuste, il faut lire le contenu des sections conditionnelles même ignorées afin de détecter les sections conditionnelles imbriquées et de s'assurer que la fin de la section conditionnelle (ignorée) la plus englobante est correctement décelée. Si une section conditionnelle marquée du mot-clé `INCLUDE` se trouve au sein d'une section conditionnelle plus importante qui elle est marqué d'un mot-clé `IGNORE`, les sections intérieure et extérieure sont toutes deux ignorées.

Si le mot-clé d'une section conditionnelle est un appel d'entité paramètre, on remplacera l'entité paramètre par son contenu avant que le processeur ne décide s'il doit inclure ou ignorer la section conditionnelle.

Exemple :

```
<!ENTITY % Ébauche 'INCLUDE' >
<!ENTITY % BonÀTirer 'IGNORE' >

<![%Ébauche;[
<!ELEMENT livre (commentaires*, titre, corps, annexes?)>
]]>
<![%BonÀTirer;[
<!ELEMENT livre (titre, corps, annexes?)>
]]>
```

4. Structures physiques

Un document XML peut être constitué d'une ou plusieurs unités de stockage. Ces unités sont appelées **entités** ; à chacune d'entre elles on associe un **contenu** et un **nom** qui les identifie, à l'exception de l'entité document (cf. ci-dessous) et du sous-ensemble externe de la DTD. Chaque document XML possède une entité appelée l'entité document qui sert de point de départ pour le processeur XML et qui peut contenir le document au complet.

Une entité peut être analysable ou non. On désigne le contenu d'une **entité analysable** sous le nom de texte de remplacement ; ce texte fait partie intégrante du document.

Une **entité non-analysable** est une ressource dont le contenu n'est pas nécessairement textuel (et pas nécessairement du texte XML même si textuel). On associe à chaque entité non-analysable une notation, identifiée par un nom. XML n'impose aucune contrainte au contenu des entités non-analysables, si ce n'est que les noms des entités et des notations soient mis à la disposition de l'application.

Les entités analysables sont appelées par leur nom au moyen d'un appel d'entité ; le nom des entités non-analysables est fourni par la valeur d'un attribut de type `ENTITY` ou `ENTITIES`.

Les **entités générales** sont destinées à être utilisées dans le contenu du document. Les **entités paramètres** sont des entités analysables destinées à être utilisées dans la DTD. Les appels à ces deux types d'entités sont différents et sont reconnus dans des contextes différents. De plus, les deux types occupent des espaces de noms distincts ; une entité paramètre et une entité générale de même nom sont deux entités distinctes. Dans ce texte, les entités générales sont parfois appelées simplement *entités* quand il n'y a pas de risque d'ambiguïté.

Note that for reliable parsing, the contents of even ignored conditional sections must be read in order to detect nested conditional sections and ensure that the end of the outermost (ignored) conditional section is properly detected. If a conditional section with a keyword of `INCLUDE` occurs within a larger conditional section with a keyword of `IGNORE`, both the outer and the inner conditional sections are ignored.

If the keyword of the conditional section is a parameter-entity reference, the parameter entity must be replaced by its content before the processor decides whether to include or ignore the conditional section.

An example:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4. Physical Structures

An XML document may consist of one or many storage units. These are called **entities**; they all have **content** and are all (except for the document entity, see below, and the external DTD subset) identified by **name**. Each XML document has one entity called the document entity, which serves as the starting point for the XML processor and may contain the whole document.

Entities may be either parsed or unparsed. A **parsed entity's** contents are referred to as its replacement text; this text is considered an integral part of the document.

An **unparsed entity** is a resource whose contents may or may not be text, and if text, may not be XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.

Parsed entities are invoked by name using entity references; unparsed entities by name, given in the value of `ENTITY` or `ENTITIES` attributes.

General entities are entities for use within the document content. In this specification, general entities are sometimes referred to with the unqualified term *entity* when this leads to no ambiguity. Parameter entities are parsed entities for use within the DTD. These two types of entities use different forms of reference and are recognized in different contexts. Furthermore, they occupy different namespaces; a parameter entity and a general entity with the same name are two distinct entities.

4.1 Appels de caractère et d'entité

Un **appel de caractère** fait référence à un caractère particulier du jeu de caractères ISO/CEI 10646, par exemple un caractère inaccessible par le biais du clavier.

Appel de caractère

```
[66] AppelCar ::= '&#' [0-9]+ ';'
                | '&#x' [0-9a-fA-F]+ ';'
                                          [CF : Caractère admissible]
```

Contrainte de forme : Caractère admissible
Le caractère faisant l'objet d'un appel de caractère doit respecter la production Car.

Si un appel de caractère commence par « `&#x` », les chiffres et lettres jusqu'au terminateur « `;` » constituent une représentation hexadécimale de la position de code du caractère dans l'ISO/CEI 10646. S'il commence seulement par « `&#` », les chiffres jusqu'au terminateur « `;` » constituent une représentation décimale de cette position de code.

Un **appel d'entité** fait référence au contenu d'une entité nommée. Les appels à des entités générales analysables utilisent l'esperluette (`&`) et le point-virgule (`;`) comme délimiteurs. Les **appels d'entités paramètres** utilisent le symbole pour cent (`%`) et le point-virgule (`;`) comme délimiteurs.

Appel d'entité

```
[67]       Appel ::= AppelEntité | AppelCar
[68] AppelEntité ::= '&' Nom ';'                  [CF : Entité déclarée]
                                                  [CV : Entité déclarée]
                                               [CF : Entité analysable]
                                                [CF : Pas de récursion]
[69]     AppelEP ::= '%' Nom ';'                  [CV : Entité déclarée]
                                                [CF : Pas de récursion]
                                                     [CF : Dans la DTD]
```

Contrainte de forme : Entité déclarée
Dans un document sans DTD, un document avec seulement un sous-ensemble interne de DTD sans appel d'entité paramètre ou un document marqué « `standalone='yes'` », le `Nom` donné dans l'appel doit correspondre au nom d'une déclaration d'entité. Toutefois, dans les documents bien formés, les entités suivantes n'ont pas à être déclarées : `amp`, `lt`, `gt`, `apos`, `quot`. La déclaration d'une entité paramètre doit précéder tout appel à celle-ci. De même, la déclaration d'une entité générale doit précéder tout appel à celle-ci apparaissant dans une valeur implicite dans une déclaration de liste d'attributs. Note : si des entités sont déclarées dans le sous-ensemble externe de DTD ou dans des entités paramètres externes, un processeur non-validateur n'est pas obligé de lire et de traiter leurs déclarations ; pour de tels documents, la règle voulant qu'une entité soit déclarée n'est une contrainte de forme que si standalone='yes'.

4.1 Character and Entity References

A **character reference** refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.

Character Reference

```
[66] CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'
                                                [WFC: Legal Character]
```

Well-Formedness Constraint: Legal Character
Characters referred to using character references must match the production for Char.

If the character reference begins with "`&#x`", the digits and letters up to the terminating `;` provide a hexadecimal representation of the character's code point in ISO/IEC 10646. If it begins just with "`&#`", the digits up to the terminating `;` provide a decimal representation of the character's code point.

An **entity reference** refers to the content of a named entity. References to parsed general entities use ampersand (`&`) and semicolon (`;`) as delimiters. **Parameter-entity references** use percent-sign (`%`) and semicolon (`;`) as delimiters.

Entity Reference

```
[67]   Reference ::= EntityRef | CharRef
[68]   EntityRef ::= '&' Name ';'                  [WFC : Entity Declared]
                                                      [VC: Entity Declared]
                                                      [WFC: Parsed Entity]
                                                      [WFC: No Recursion]
[69] PEReference ::= '%' Name ';'                   [VC : Entity Declared]
                                                    [WFC: No Recursion]
                                                          [WFC: In DTD]
```

Well-Formedness Constraint: Entity Declared
In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with "`standalone='yes'`", the `Name` given in the entity reference must match that in an entity declaration, except that well-formed documents need not declare any of the following entities: `amp`, `lt`, `gt`, `apos`, `quot`. The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration. Note that if entities are declared in the external subset or in external parameter entities, a non-validating processor is not obligated to read and process their declarations; for such documents, the rule that an entity must be declared is a well-formedness constraint only if standalone='yes'.

Contrainte de validité : Entité déclarée
Dans un document avec sous-ensemble externe de DTD ou des entités paramètres externes avec « `standalone='no'` », le `Nom` donné dans l'appel doit correspondre au nom d'une déclaration d'entité. À des fins d'interopérabilité, les documents valides devraient déclarer les entités `amp`, `lt`, `gt`, `apos`, `quot`, sous la forme précisée en « 4.6 Entités prédéfinies ». La déclaration d'une entité paramètre doit précéder tout appel à celle-ci. De même, la déclaration d'une entité générale doit précéder tout appel à celle-ci apparaissant dans une valeur implicite dans une déclaration de liste d'attributs.

Contrainte de forme : Entité analysable
Un appel d'entité ne doit pas contenir le nom d'une entité non-analysable. On ne peut faire référence à des entités non-analysables que par le biais de valeurs d'attribut déclarées de type `ENTITY` ou `ENTITIES`.

Contrainte de forme : Pas de récursion
Une entité analysable ne doit pas contenir d'appel récursif à elle-même, directement ou indirectement.

Contrainte de forme : Dans la DTD
Les appels d'entité paramètre ne peuvent se trouver que dans la DTD.

Exemples d'appels de caractère et d'entité :

```
Tapez <touche>plus-petit-que</touche> (&#x3C;) pour sauvegarder les options.
Ce document a été rédigé le &datedoc; et
est classé &niveau-de-sécurité;.}
```

Exemple d'appel d'entité paramètre :

```
<!-- déclaration de l'entité paramètre "ISOLat2"... -->
<!ENTITY % ISOLat2
         SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... et appel -->
%ISOLat2;
```

4.2 Déclarations d'entités

On déclare des entités de la façon suivante :

Déclaration d'entité

```
[70] DéclEntité ::= DéclEG | DéclEP
[71]     DéclEG ::= '<!ENTITY' S Nom S DéfEntité S? '>'
[72]     DéclEP ::= '<!ENTITY' S '%' S Nom S DéfEP S? '>'
[73]  DéfEntité ::= ValeurEntité | (IdExterne DéclNdata?)
[74]      DéfEP ::= ValeurEntité | IdExterne
```

Le `Nom` identifie l'entité dans un appel d'entité ou, dans le cas d'une entité non-analysable, dans la valeur d'un attribut de type `ENTITY` ou `ENTITIES`. Si une même entité est déclarée plus d'une fois, seule la première déclaration compte ; un processeur XML peut alors émettre, au gré de l'utilisateur, un avertissement de déclarations multiples.

Validity Constraint: Entity Declared
In a document with an external subset or external parameter entities with "`standalone='no'`", the `Name` given in the entity reference must match that in an entity declaration. For interoperability, valid documents should declare the entities `amp`, `lt`, `gt`, `apos`, `quot`, in the form specified in "4.6 Predefined Entities". The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration.

Well-Formedness Constraint: Parsed Entity
An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referred to only in attribute values declared to be of type `ENTITY` or `ENTITIES`.

Well-Formedness Constraint: No Recursion
A parsed entity must not contain a recursive reference to itself, either directly or indirectly.

Well-Formedness Constraint: In DTD
Parameter-entity references may only appear in the DTD.

Examples of character and entity references:

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.}
```

Example of a parameter-entity reference:

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2
         SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

4.2 Entity Declarations

Entities are declared thus:

Entity Declaration

```
[70] EntityDecl ::= GEDecl | PEDecl
[71]     GEDecl ::= '<!ENTITY' S Name S EntityDef S? '>'
[72]     PEDecl ::= '<!ENTITY' S '%' S Name S PEDef S? '>'
[73]  EntityDef ::= EntityValue | (ExternalID NDataDecl?)
[74]      PEDef ::= EntityValue | ExternalID
```

The `Name` identifies the entity in an entity reference or, in the case of an unparsed entity, in the value of an `ENTITY` or `ENTITIES` attribute. If the same entity is declared more than once, the first declaration encountered is binding; at user option, an XML processor may issue a warning if entities are declared multiple times.

4.2.1 Entités internes Si la définition de l'entité est une `ValeurEntité`, l'entité ainsi définie est appelée **entité interne**. Il n'y a pas d'objet stocké séparément et le contenu de l'entité est précisé dans la déclaration. Remarquons que la production du texte de remplacement peut nécessiter le traitement d'appels de caractère et d'entité dans la valeur littérale d'entité : cf. « 4.5 Construction du texte de remplacement d'une entité interne ».

Une entité interne est une entité analysable.

Exemple de déclaration d'entité interne :

```
<!ENTITY Avancement "Ceci est une version prépublication
 de la spécification.">
```

4.2.2 Entités externes Si l'entité n'est pas interne, il s'agit d'une **entité externe**, déclarée comme suit :

Déclaration d'entité externe

```
[75] IdExterne ::= 'SYSTEM' S LittéralSystème
                 | 'PUBLIC' S IdPubLittéral S LittéralSystème
[76] DéclNdata ::= S 'NDATA' S Nom              [CV : Notation déclarée]
```

Si la production `DéclNdata` est présente, il s'agit d'une entité non-analysable générale ; autrement il s'agit d'une entité analysable.

Contrainte de validité : Notation déclarée
Le `Nom` doit correspondre au nom d'une notation déclarée.

La production `LittéralSystème` est appelée **identificateur système** de l'entité. Il s'agit d'un URI, qui peut être utilisé pour récupérer l'entité. Il est à noter que le croisillon (`#`) et l'identificateur de fragment, qui sont fréquemment utilisés avec des URI, ne font pas formellement partie de l'URI. Un processeur XML peut signaler une erreur si un identificateur de système contient un identificateur de fragment. Les URI relatifs doivent être interprétés par rapport à l'emplacement de la ressource au sein de laquelle la déclaration d'entité se trouve, sauf en cas d'information de provenance externe et qui ne relève pas de ce standard, comme un élément XML spécial défini dans une DTD particulière ou une instruction de traitement définie par une application particulière. Un URI peut donc être relatif à l'entité document, à l'entité contenant le sous-ensemble externe de DTD ou à une quelconque entité paramètre externe.

Un processeur XML devrait traiter un caractère non-ASCII dans un URI en le représentant sous forme d'un ou plusieurs octets en UTF-8, puis en transformant chacun de ces octets selon le mécanisme de transformation URI (c'est à dire en convertissant chaque octet en %HH, où HH est la notation hexadécimale de la valeur de l'octet).

En plus d'un identificateur système, un identificateur externe peut contenir un **identificateur public**. Un processeur XML qui essaie de récupérer le contenu d'une entité peut utiliser l'identificateur public pour en tirer un URI de remplacement. Si le processeur en est incapable, il doit utiliser l'URI précisé par l'identificateur système. L'identificateur public doit être normalisé en transformant toute suite de blancs en un seul caractère espace (#x20) et en éliminant tout blanc de tête et de queue.

4.2.1 Internal Entities If the entity definition is an `EntityValue`, the defined entity is called an **internal entity**. There is no separate physical storage object, and the content of the entity is given in the declaration. Note that some processing of entity and character references in the literal entity value may be required to produce the correct replacement text: see "4.5 Construction of Internal Entity Replacement Text".

An internal entity is a parsed entity.

Example of an internal entity declaration:

```
<!ENTITY Pub-Status "This is a pre-release of the
  specification.">
```

4.2.2 External Entities If the entity is not internal, it is an **external entity**, declared as follows:

External Entity Declaration

```
[75] ExternalID ::= 'SYSTEM' S SystemLiteral
                  | 'PUBLIC' S PubidLiteral S SystemLiteral
[76]  NDataDecl ::= S 'NDATA' S Name              [VC: Notation Declared]
```

If the `NDataDecl` is present, this is a general unparsed entity; otherwise it is a parsed entity.

Validity Constraint: Notation Declared
The `Name` must match the declared name of a notation.

The `SystemLiteral` is called the entity's **system identifier**. It is a URI, which may be used to retrieve the entity. Note that the hash mark (`#`) and fragment identifier frequently used with URIs are not, formally, part of the URI itself; an XML processor may signal an error if a fragment identifier is given as part of a system identifier. Unless otherwise provided by information outside the scope of this specification (e.g. a special XML element type defined by a particular DTD, or a processing instruction defined by a particular application specification), relative URIs are relative to the location of the resource within which the entity declaration occurs. A URI might thus be relative to the document entity, to the entity containing the external DTD subset, or to some other external parameter entity.

An XML processor should handle a non-ASCII character in a URI by representing the character in UTF-8 as one or more bytes, and then escaping these bytes with the URI escaping mechanism (i.e., by converting each byte to %HH, where HH is the hexadecimal notation of the byte value).

In addition to a system identifier, an external identifier may include a **public identifier**. An XML processor attempting to retrieve the entity's content may use the public identifier to try to generate an alternative URI. If the processor is unable to do so, it must use the URI specified in the system literal. Before a match is attempted, all strings of white space in the public identifier must be normalized to single space characters (#x20), and leading and trailing white space must be removed.

Exemples de déclaration d'entité externe :

```
<!ENTITY écoutille-ouverte
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY écoutille-ouverte
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY image-écoutille
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3 Entités analysables

4.3.1 La déclaration de texte Les entités analysables externes peuvent commencer par une **déclaration de texte**.

Déclaration de texte

```
[77] DéclTexte ::= '<?xml' InfoVersion? DéclCodage S? '?>'
```

La déclaration de texte doit être fournie littéralement et non pas par appel d'une entité analysable. Elle ne peut se trouver qu'au début d'une entité analysable externe.

4.3.2 Entités analysables bien formées On dit d'une entité document qu'elle est bien formée si elle correspond à la production `document`. Une entité générale analysable externe est bien formée si elle correspond à la production `entAnalExt`. Une entité paramètre externe est bien formée si elle correspond à la production `entParExt`.

Entité analysable externe bien formée

```
[78] entAnalExt ::= DéclTexte?  contenu
[79]  entParExt ::= DéclTexte?  déclSousEnsembleExt
```

Une entité générale analysable interne est bien formée si son texte de remplacement correspond à la production `contenu`. Toute entité paramètre interne est bien formée par définition.

Il découle du caractère bien formé des entités que les structures logique et physique d'un document XML s'imbriquent proprement ; aucun élément, commentaire, balise ouvrante, balise fermante, balise d'élément vide, instruction de traitement, appel de caractère ou appel d'entité ne peut commencer dans une entité et se terminer dans une autre.

4.3.3 Codage des caractères dans les entités Chaque entité analysable externe d'un document XML peut utiliser un codage différent pour ses caractères. Tous les processeurs XML doivent être à même de lire les entités en UTF-8 ou UTF-16.

Examples of external entity declarations:

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3 Parsed Entities

4.3.1 The Text Declaration External parsed entities may each begin with a **text declaration**.

Text Declaration

```
[77] TextDecl ::= '<?xml' VersionInfo? EncodingDecl S? '?>'
```

The text declaration must be provided literally, not by reference to a parsed entity. No text declaration may appear at any position other than the beginning of an external parsed entity.

4.3.2 Well-Formed Parsed Entities The document entity is well-formed if it matches the production labeled `document`. An external general parsed entity is well-formed if it matches the production labeled `extParsedEnt`. An external parameter entity is well-formed if it matches the production labeled `extPE`.

Well-Formed External Parsed Entity

```
[78] extParsedEnt ::= TextDecl?  content
[79]        extPE ::= TextDecl?  extSubsetDecl
```

An internal general parsed entity is well-formed if its replacement text matches the production labeled `content`. All internal parameter entities are well-formed by definition.

A consequence of well-formedness in entities is that the logical and physical structures in an XML document are properly nested; no start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another.

4.3.3 Character Encoding in Entities Each external parsed entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in either UTF-8 or UTF-16.

Les entités codées en UTF-16 doivent commencer par une marque d'ordre d'octets telle que décrite par l'annexe E de l'ISO/CEI 10646 ou par l'annexe B d'Unicode (le caractère ESPACE INSÉCABLE SANS CHASSE, #xFEFF). La marque est une signature de codage et ne fait partie ni du balisage ni des données textuelles du document XML. Les processeurs XML doivent être capables d'utiliser ce caractère pour distinguer les documents codés en UTF-8 et en UTF-16.

Bien que seul le soutien des codages UTF-8 et UTF-16 ne soit requis des processeurs XML, il est clair que d'autres codages sont couramment utilisés et qu'il peut être souhaitable que les processeurs XML puissent lire les entités qui les emploient. Les entités analysables stockées dans un codage autre qu'UTF-8 ou UTF-16 doivent commencer par une déclaration de texte contenant une déclaration de codage :

Déclaration de codage

```
[80] DéclCodage ::= S 'encoding' Égal
                    ('"' NomCodage '"' |  "'" NomCodage "'" )
[81]  NomCodage ::= [A-Za-z] ([A-Za-z0-9._] | '-')*
              /* Nom de codage ne contenant que des caractères latins */
```

Dans l'entité document, la déclaration de codage se trouve dans la déclaration XML. `NomCodage` est le nom du codage utilisé.

Dans une déclaration de codage, les valeurs « `UTF-8` », « `UTF-16` », « `ISO-10646-UCS-2` » et « `ISO-10646-UCS-4` » devraient être utilisées pour les divers codages et transformation d'Unicode / ISO/CEI 10646, les valeurs « `ISO-8859-1` », « `ISO-8859-2` », ... « `ISO-8859-9` » pour les parties de l'ISO 8859 et les valeurs « `ISO-2022-JP` », « `Shift_JIS` » et « `EUC-JP` » pour les diverses formes de codage de JIS X-0208-1997. Les processeurs XML peuvent reconnaître d'autres codages ; il est recommandé d'utiliser, pour les codages de caractères répertoriés (comme *charset*) par l'Internet Assigned Numbers Authority [IANA] autres que ceux ci-dessus, les noms du répertoire de l'IANA. Il est à noter que ces noms sont définis comme indépendants de la casse ; les correspondances avec ces noms doivent donc être faites sans égard à la casse.

En l'absence d'information donnée par un protocole de transport externe (p. ex. HTTP ou MIME), il est erroné de présenter au processeur XML une entité qui, comportant une déclaration de codage, serait codée autrement, de faire apparaître une déclaration de codage ailleurs qu'au début d'une entité externe ou de coder autrement qu'en UTF-8 une entité qui ne commence ni par une *marque d'ordre d'octets* ni par une déclaration de codage. L'ASCII étant un sous-ensemble d'UTF-8, les entités codées en ASCII n'ont pas absolument besoin de déclaration de codage.

Une erreur fatale doit être signalée quand un processeur XML se trouve en présence d'une entité utilisant un codage que le processeur est incapable de traiter.

Exemples de déclaration de codage :

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

Entities encoded in UTF-16 must begin with the Byte Order Mark described by ISO/IEC 10646 Annex E and Unicode Appendix B (the ZERO WIDTH NO-BREAK SPACE character, #xFEFF). This is an encoding signature, not part of either the markup or the character data of the XML document. XML processors must be able to use this character to differentiate between UTF-8 and UTF-16 encoded documents.

Although an XML processor is required to read only entities in the UTF-8 and UTF-16 encodings, it is recognized that other encodings are used around the world, and it may be desired for XML processors to read entities that use them. Parsed entities which are stored in an encoding other than UTF-8 or UTF-16 must begin with a text declaration containing an encoding declaration:

Encoding Declaration

```
[80] EncodingDecl ::= S 'encoding' Eq
                      ('"' EncName '"' | "'" EncName "'" )
[81]      EncName ::= [A-Za-z] ([A-Za-z0-9._] | '-')*
                       /* Encoding name contains only Latin characters */
```

In the document entity, the encoding declaration is part of the XML declaration. The `EncName` is the name of the encoding used.

In an encoding declaration, the values `"UTF-8"`, `"UTF-16"`, `"ISO-10646-UCS-2"`, and `"ISO-10646-UCS-4"` should be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values `"ISO-8859-1"`, `"ISO-8859-2"`, ... `"ISO-8859-9"` should be used for the parts of ISO 8859, and the values `"ISO-2022-JP"`, `"Shift_JIS"`, and `"EUC-JP"` should be used for the various encoded forms of JIS X-0208-1997. XML processors may recognize other encodings; it is recommended that character encodings registered (as *charsets*) with the Internet Assigned Numbers Authority [IANA], other than those just listed, should be referred to using their registered names. Note that these registered names are defined to be case-insensitive, so processors wishing to match against them should do so in a case-insensitive way.

In the absence of information provided by an external transport protocol (e.g. HTTP or MIME), it is an error for an entity including an encoding declaration to be presented to the XML processor in an encoding other than that named in the declaration, for an encoding declaration to occur other than at the beginning of an external entity, or for an entity which begins with neither a Byte Order Mark nor an encoding declaration to use an encoding other than UTF-8. Note that since ASCII is a subset of UTF-8, ordinary ASCII entities do not strictly need an encoding declaration.

It is a fatal error when an XML processor encounters an entity with an encoding that it is unable to process.

Examples of encoding declarations:

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

4.4 Traitement des entités et des appels par un processeur XML

Le tableau ci-dessous résume dans quels contextes les appels de caractère, les appels d'entité et les références à des entités non-analysables peuvent se produire ainsi que le comportement exigé d'un processeur XML dans chaque cas. Les étiquettes de la colonne de gauche indiquent le contexte :

Appel dans le contenu Un appel n'importe où entre la balise ouvrante et la balise fermante d'un élément ; correspond au non-terminal `contenu`.

Appel dans une valeur d'attribut Un appel soit dans la valeur d'un attribut dans une balise ouvrante, soit dans une valeur implicite dans une déclaration d'attribut ; correspond au non-terminal `ValeurAtt`.

Valeur d'attribut Un `Nom`, et non pas un appel, apparaissant soit comme valeur d'un attribut déclaré de type `ENTITY`, soit comme un des membres (délimités par des espaces) de la valeur d'un attribut déclaré de type `ENTITIES`.

Appel dans une valeur d'entité Un appel dans la valeur littérale d'entité d'une entité paramètre ou interne dans la déclaration d'entité ; correspond au non-terminal `ValeurEntité`.

Appel dans la DTD Un appel dans le sous-ensemble interne ou externe de la DTD, mais ailleurs que dans une `ValeurEntité` ou une `ValeurAtt`.

	Type d'entité				**Caractère**
	paramètre	*interne générale*	*externe analysable générale*	*non-analysable*	
Appel dans le contenu	Non reconnu	Inclus	Inclus si validation	Interdit	Inclus
Appel dans une valeur d'attribut	Non reconnu	Inclus dans littéral	Interdit	Interdit	Inclus
Appel dans une valeur d'attribut	Non reconnu	Interdit	Interdit	Signalé	Non reconnu
Appel dans une valeur d'entité	Inclus dans littéral	Non interprété	Non interprété	Interdit	Inclus
Appel dans la DTD	Inclus comme EP	Interdit	Interdit	Interdit	Interdit

4.4.1 Non reconnu Hors de la DTD, le caractère `%` n'a aucune signification particulière ; pour cette raison, ce qui serait un appel d'entité paramètre dans la DTD n'est pas reconnu comme balisage dans le `contenu`. De même, les noms des entités non-analysables ne sont pas reconnus sauf dans les valeurs d'attributs de type approprié.

4.4.2 Inclus Une entité est **incluse** quand son texte de remplacement est récupéré et traité, à la place de l'appel lui-même, comme s'il se trouvait dans le document à l'endroit où l'appel est reconnu. Le texte de remplacement peut contenir des données textuelles et du balisage (sauf des entités paramètres) qui doit être reconnu de la manière habituelle, sauf que le texte de remplacement des entités utilisées pour déguiser des délimiteurs de balisage (les entités `amp`, `lt`, `gt`, `apos`, `quot`) est toujours traité comme des données. (La chaîne « `AT&T;` » devient « `AT&T;` » et l'esperluette restante n'est pas reconnue comme un délimiteur d'appel d'entité). Un appel de caractère est **inclus** quand le caractère en question est traité à la place de l'appel lui-même.

4.4 XML Processor Treatment of Entities and References

The table below summarizes the contexts in which character references, entity references, and invocations of unparsed entities might appear and the required behavior of an XML processor in each case. The labels in the leftmost column describe the recognition context:

Reference in Content as a reference anywhere after the start-tag and before the end-tag of an element; corresponds to the nonterminal `content`.

Reference in Attribute Value as a reference within either the value of an attribute in a start-tag, or a default value in an attribute declaration; corresponds to the nonterminal `AttValue`.

Occurs as Attribute Value as a `Name`, not a reference, appearing either as the value of an attribute which has been declared as type `ENTITY`, or as one of the space-separated tokens in the value of an attribute which has been declared as type `ENTITIES`.

Reference in Entity Value as a reference within a parameter or internal entity's literal entity value in the entity's declaration; corresponds to the nonterminal `EntityValue`.

Reference in DTD as a reference within either the internal or external subsets of the DTD, but outside of an `EntityValue` or `AttValue`.

	Entity Type				**Character**
	Parameter	*Internal General*	*External Parsed General*	*Unparsed*	
Reference in Content	Not recognized	Included	Included if validating	Forbidden	Included
Reference in Attribute Value	Not recognized	Included in literal	Forbidden	Forbidden	Included
Occurs as Attribute Value	Not recognized	Forbidden	Forbidden	Notify	Not recognized
Reference in EntityValue	Included in literal	Bypassed	Bypassed	Forbidden	Included
Reference in DTD	Included as PE	Forbidden	Forbidden	Forbidden	Forbidden

4.4.1 Not Recognized Outside the DTD, the `%` character has no special significance; thus, what would be parameter entity references in the DTD are not recognized as markup in `content`. Similarly, the names of unparsed entities are not recognized except when they appear in the value of an appropriately declared attribute.

4.4.2 Included An entity is **included** when its replacement text is retrieved and processed, in place of the reference itself, as though it were part of the document at the location the reference was recognized. The replacement text may contain both character data and (except for parameter entities) markup, which must be recognized in the usual way, except that the replacement text of entities used to escape markup delimiters (the entities `amp`, `lt`, `gt`, `apos`, `quot`) is always treated as data. (The string "`AT&T;`" expands to "`AT&T;`" and the remaining ampersand is not recognized as an entity-reference delimiter.) A character reference is **included** when the indicated character is processed in place of the reference itself.

4.4.3 Inclus si validation Pour valider un document, un processeur XML doit inclure le texte de remplacement des entités analysables qu'il reconnaît. Toutefois, si une entité est externe et si le processeur ne cherche pas à valider le document, il peut ne pas inclure le texte de remplacement. Un tel processeur non-validateur doit signaler à l'application, le cas échéant, qu'il a reconnu mais n'a pas lu une entité.

Cette règle est motivée par le fait que l'inclusion automatique prévue par le mécanisme d'entité de SGML et XML, conçu avant tout pour faciliter la modularité lors de la rédaction, n'est pas nécessairement approprié pour d'autres applications. En feuilletant des documents, par exemple, on pourra choisir de donner une indication visuelle de la présence d'une entité externe et de ne la récupérer pour affichage que sur demande.

4.4.4 Interdit Les formes suivantes sont interdites et constituent des erreurs fatales :

— un appel à une entité non-analysable ;
— un appel de caractère ou un appel d'entité générale dans la DTD, sauf dans une `ValeurEntité` ou une `ValeurAtt` ;
— un appel d'entité externe dans une valeur d'attribut.

4.4.5 Inclus dans littéral Lorsqu'un appel d'entité apparaît dans une valeur d'attribut ou lorsqu'un appel d'entité paramètre apparaît dans une valeur littérale d'entité, son texte de remplacement est traité à la place de l'appel comme s'il se trouvait dans le document au lieu où l'appel est reconnu, sauf que les caractères guillemet simple et double dans le texte de remplacement sont toujours traités comme données et ne terminent pas le littéral. L'exemple qui suit est bien formé :

```
<!ENTITY % ON '"Oui"' >
<!ENTITY CeQuilDit "Il dit &ON;" >
```

alors que celui-ci ne l'est pas :

```
<!ENTITY FinAttr "27'" >
<élément attribut='a-&FinAttr;>
```

4.4.6 Signalé Quand le nom d'une entité non-analysable fait partie de la valeur d'un attribut déclaré de type `ENTITY` ou `ENTITIES`, un processeur validateur doit informer l'application des identificateurs système et public (le cas échéant) de l'entité et de sa notation.

4.4.7 Non interprété Lorsqu'un appel d'entité générale se trouve dans une `ValeurEntité` dans une déclaration d'entité, il n'est pas interprété et est laissé tel quel.

4.4.8 Inclus comme EP (entité paramètre) De même que les entités analysables externes, les entités paramètres n'ont à être incluse que s'il y a validation. Lorsqu'un appel d'entité paramètre est reconnu dans la DTD et inclus, son texte de remplacement est augmenté d'une espace (#x20) au début et à la fin ; ceci dans le but de s'assurer que le texte de remplacement des entités paramètres contient bien un nombre entier d'unités lexicales dans la DTD.

4.4.3 Included If Validating When an XML processor recognizes a reference to a parsed entity, in order to validate the document, the processor must include its replacement text. If the entity is external, and the processor is not attempting to validate the XML document, the processor may, but need not, include the entity's replacement text. If a non-validating parser does not include the replacement text, it must inform the application that it recognized, but did not read, the entity.

This rule is based on the recognition that the automatic inclusion provided by the SGML and XML entity mechanism, primarily designed to support modularity in authoring, is not necessarily appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external parsed entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand.

4.4.4 Forbidden The following are forbidden, and constitute fatal errors:

- the appearance of a reference to an unparsed entity.
- the appearance of any character or general-entity reference in the DTD except within an `EntityValue` or `AttValue`.
- a reference to an external entity in an attribute value.

4.4.5 Included in Literal When an entity reference appears in an attribute value, or a parameter entity reference appears in a literal entity value, its replacement text is processed in place of the reference itself as though it were part of the document at the location the reference was recognized, except that a single or double quote character in the replacement text is always treated as a normal data character and will not terminate the literal. For example, this is well-formed:

```
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "He said &YN;" >}
```

while this is not:

```
<!ENTITY EndAttr "27'" >
<element attribute='a-&EndAttr;>}
```

4.4.6 Notify When the name of an unparsed entity appears as a token in the value of an attribute of declared type `ENTITY` or `ENTITIES`, a validating processor must inform the application of the system and public (if any) identifiers for both the entity and its associated notation.

4.4.7 Bypassed When a general entity reference appears in the `EntityValue` in an entity declaration, it is bypassed and left as is.

4.4.8 Included as PE Just as with external parsed entities, parameter entities need only be included if validating. When a parameter-entity reference is recognized in the DTD and included, its replacement text is enlarged by the attachment of one leading and one following space (#x20) character; the intent is to constrain the replacement text of parameter entities to contain an integral number of grammatical tokens in the DTD.

4.5 Construction du texte de remplacement d'une entité interne

En faisant état du traitement des entités internes, il convient de distinguer deux formes de la valeur de l'entité. La **valeur littérale d'entité** est la chaîne entre guillemets qu'on trouve dans la déclaration d'entité, correspondant au non-terminal `ValeurEntité`. Le **texte de remplacement** est le contenu de l'entité, après remplacement des appels de caractère et d'entités paramètres.

La valeur littérale d'entité telle que donnée dans une déclaration d'entité interne (`ValeurEntité`) peut contenir des appels de caractère, d'entité paramètre et d'entité générale. Ces appels doivent être complètement contenus dans la valeur littérale d'entité. Le texte de remplacement qui est inclus, tel que décrit ci-dessus, doit contenir le *texte de remplacement* de tout appel d'entité paramètre ainsi que le caractère correspondant à tout appel de caractère ; toutefois, les appels d'entité générale doivent être laissés tels quels, sans remplacement. Par exemple, étant donné les déclarations suivantes :

```
<!ENTITY % éd   "&#xc9;ditions Gallimard">
<!ENTITY droits "Tous droits réservés">
<!ENTITY livre  "La Peste : Albert Camus, &#xA9; 1947 %éd;. &droits;">
```

le texte de remplacement de l'entité « `livre` » est :

```
La Peste : Albert Camus,
l' 1947 Éditions Gallimard. &droits;
```

L'appel d'entité générale « `&droits;` » sera remplacé si l'appel « `&livre;` » venait à apparaître dans le contenu du document ou dans une valeur d'attribut.

Ces règles simples peuvent avoir des interactions complexes ; on trouvera un exemple difficile expliqué en détails en « D. Remplacement des appels d'entité et de caractère ».

4.6 Entités prédéfinies

On peut employer des appels d'entité ou de caractère pour **déguiser** le signe inférieur-à, l'esperluette et d'autres délimiteurs. Un jeu d'entités générales (`amp`, `lt`, `gt`, `apos`, `quot`) est défini à cette fin. On peut aussi employer des appels numériques de caractère ; ils sont remplacés dès que reconnus et doivent être traités comme données textuelles, de sorte que les appels numériques de caractère « `&#60;` » et « `&#38;` » peuvent être utilisés pour déguiser `<` et `&` dans des données textuelles.

Tous les processeurs XML doivent reconnaître ces entités, qu'elles soient déclarées ou non. À des fins d'interopérabilité, les documents XML valides devraient déclarer ces entités, comme n'importe quelles autres, avant tout appel. S'il y a lieu, les entités en question devraient être déclarées comme entités internes dont le texte de remplacement est le caractère à déguiser ou un appel de caractère à ce caractère, comme ci-dessous :

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

4.5 Construction of Internal Entity Replacement Text

In discussing the treatment of internal entities, it is useful to distinguish two forms of the entity's value. The **literal entity value** is the quoted string actually present in the entity declaration, corresponding to the non-terminal `EntityValue`. The **replacement text** is the content of the entity, after replacement of character references and parameter-entity references.

The literal entity value as given in an internal entity declaration (`EntityValue`) may contain character, parameter-entity, and general-entity references. Such references must be contained entirely within the literal entity value. The actual replacement text that is included as described above must contain the *replacement text* of any parameter entities referred to, and must contain the character referred to, in place of any character references in the literal entity value; however, general-entity references must be left as-is, unexpanded. For example, given the following declarations:

```
<!ENTITY % pub  "&#xc9;ditions Gallimard">
<!ENTITY rights "All rights reserved">
<!ENTITY book   "La Peste: Albert Camus, &#xA9; 1947 %pub;. &rights;">
```

then the replacement text for the entity "`book`" is:

```
La Peste: Albert Camus,
l' 1947 Éditions Gallimard. &rights;
```

The general-entity reference "`&rights;`" would be expanded should the reference "`&book;`" appear in the document's content or an attribute value.

These simple rules may have complex interactions; for a detailed discussion of a difficult example, see "D. Expansion of Entity and Character References".

4.6 Predefined Entities

Entity and character references can both be used to **escape** the left angle bracket, ampersand, and other delimiters. A set of general entities (`amp`, `lt`, `gt`, `apos`, `quot`) is specified for this purpose. Numeric character references may also be used; they are expanded immediately when recognized and must be treated as character data, so the numeric character references "`&#60;`" and "`&#38;`" may be used to escape `<` and `&` when they occur in character data.

All XML processors must recognize these entities whether they are declared or not. For interoperability, valid XML documents should declare these entities, like any others, before using them. If the entities in question are declared, they must be declared as internal entities whose replacement text is the single character being escaped or a character reference to that character, as shown below.

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

Remarquons que les caractères < et & dans les déclarations de « lt » et « amp »sont déguisés doublement, de manière à satisfaire l'exigence que le remplacement d'entité soit bien formé.

4.7 Déclarations de notation

Une **notation** désigne par un nom le format d'une entité non-analysable, le format d'un élément muni d'un attribut notation ou l'application cible d'une instruction de traitement.

Une **déclaration de notation** donne un nom à la notation, nom servant dans les déclarations d'entité et de liste d'attributs et dans les stipulations d'attribut, ainsi qu'un identificateur externe qui peut permettre à un processeur XML ou à son application cliente de repérer un assistant capable de traiter des données en cette notation.

Déclaration de notation

```
[82] DéclNotation ::= '<!NOTATION' S Nom S (IdExterne | IdPublic) S? '>'
[83]      IdPublic ::= 'PUBLIC' S IdPubLittéral
```

Les processeurs XML doivent fournir aux applications les nom et identificateur externe de toute notation déclarée et mentionnée dans une valeur d'attribut, une définition d'attribut ou une déclaration d'entité. Le processeur XML peut aussi transformer l'identificateur externe en un identificateur système, un nom de fichier ou toute autre information utile à l'application pour appeler un processeur propre à la notation. (Il n'y a pas erreur, toutefois, lorsqu'un document XML déclare et mentionne des notations pour lesquelles des processeurs idoines ne sont pas disponibles sur le système où tourne le processeur XML ou l'application).

4.8 L'entité document

L'entité document sert de racine à l'arbre des entités et de point de départ au processeur XML. Ce standard ne précise pas comment un processeur XML peut localiser l'entité document. Contrairement aux autres entités, l'entité document n'a pas de nom et peut fort bien apparaître à l'entrée d'un processeur sans la moindre identification.

5. Conformité

5.1 Processeurs validateurs et non-validateurs

Les processeurs XML conformes sont de deux classes : validateur et non-validateur.

Les processeurs des deux classes doivent signaler les violations de contraintes de forme de ce standard dans le contenu de l'entité document et de toute autre entité analysable qu'ils lisent.

Les **processeurs validateurs** doivent en outre signaler les violations de contraintes exprimées par les déclarations de la DTD et les violations des contraintes de validité précisées par ce standard. Pour ce faire, les processeurs XML validateurs doivent lire et traiter la DTD au complet ainsi que toutes les entités analysables externes mentionnées dans le document.

Note that the < and & characters in the declarations of "lt" and "amp" are doubly escaped to meet the requirement that entity replacement be well-formed.

4.7 Notation Declarations

Notations identify by name the format of unparsed entities, the format of elements which bear a notation attribute, or the application to which a processing instruction is addressed.

Notation declarations provide a name for the notation, for use in entity and attribute-list declarations and in attribute specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.

Notation Declarations

```
[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID | PublicID) S? '>'
[83]     PublicID ::= 'PUBLIC' S PubidLiteral
```

XML processors must provide applications with the name and external identifier(s) of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They may additionally resolve the external identifier into the system identifier, file name, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

4.8 Document Entity

The **document entity** serves as the root of the entity tree and a starting-point for an XML processor. This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity has no name and might well appear on a processor input stream without any identification at all.

5. Conformance

5.1 Validating and Non-Validating Processors

Conforming XML processors fall into two classes: validating and non-validating.

Validating and non-validating processors alike must report violations of this specification's well-formedness constraints in the content of the document entity and any other parsed entities that they read.

Validating processors must report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification. To accomplish this, validating XML processors must read and process the entire DTD and all external parsed entities referenced in the document.

Les processeurs non-validateurs ne sont tenus que de vérifier la forme de l'entité document, y compris le sous-ensemble interne de la DTD au complet. Bien qu'ils ne soient pas tenus de vérifier la validité du document, ils doivent **traiter** toutes les déclarations lues dans le sous-ensemble interne de la DTD ainsi que dans toute entité paramètre lue, jusqu'au premier appel d'entité paramètre qui n'est *pas* lu ; **traiter** signifie qu'ils doivent utiliser ces déclarations pour normaliser les valeurs d'attribut, inclure le texte de remplacement des entités internes et fournir les valeurs implicites d'attribut. Ils ne doivent ni traiter les déclarations d'entité ni les déclarations de liste d'attributs qui suivent un appel d'entité paramètre qui n'a pas été lu, puisque cette entité pourrait contenir des déclarations contradictoires.

5.2 Utilisation des processeurs XML

Le comportement d'un processeur XML validateur est éminemment prévisible : il doit lire toutes les parties d'un document et signaler toute violation des contraintes de forme et de validité. On est moins exigeant des processeurs non-validateurs, qui ne sont tenus de lire que l'entité document. Il en résulte deux effets qui peuvent être d'importance pour les utilisateurs de processeurs XML :

— Certaines erreurs de forme, notamment celles dont la détection exige de lire les entités externes, peuvent échapper à un processeur non-validateur. Des exemples de contraintes pouvant être violées sans avertissement sont celles appelées Entité déclarée, Entité analysable et Pas de récursion, ainsi que certains cas décrits comme Interdit dans « 4.4 Traitement des entités et des appels par un processeur XML ».
— L'information transmise du processeur à l'application peut varier, selon que le processeur lit ou non les entités paramètres et externes. Ainsi, un processeur non-validateur peut ne pas normaliser des valeurs d'attribut, ne pas inclure le texte de remplacement d'entités internes ou ne pas fournir des valeurs implicites d'attribut, lorsque ces actions impliquent la lecture préalable de déclarations situées dans des entités paramètres ou externes.

Afin d'optimiser l'interopérabilité entre différents processeurs XML, les applications utilisant des processeurs non-validateurs ne devraient pas se fier à des comportements facultatifs de la part de ces processeurs. Les applications qui nécessitent des fonctions comme les valeurs implicites d'attribut ou des entités internes déclarées dans des entités externes devraient utiliser des processeurs validateurs.

6. Notation

La grammaire formelle de XML est précisée dans ce standard en utilisant une notation simple de forme Backus-Naur étendue (EBNF). Chaque règle de la grammaire définit un symbole, sous la forme :

```
symbole ::= expression
```

Les symboles prennent une majuscule initiale si une expression régulière les définit, une minuscule initiale autrement. Les chaînes littérales sont entre guillemets anglais simples ou doubles ('chaîne littérale', "chaîne littérale").

Non-validating processors are required to check only the document entity, including the entire internal DTD subset, for well-formedness. While they are not required to check the document for validity, they are required to **process** all the declarations they read in the internal DTD subset and in any parameter entity that they read, up to the first reference to a parameter entity that they do *not* read; that is to say, they must use the information in those declarations to normalize attribute values, include the replacement text of internal entities, and supply default attribute values. They must not processentity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations.

5.2 Using XML Processors

The behavior of a validating XML processor is highly predictable; it must read every piece of a document and report all well-formedness and validity violations. Less is required of a non-validating processor; it need not read any part of the document other than the document entity. This has two effects that may be important to users of XML processors:

- Certain well-formedness errors, specifically those that require reading external entities, may not be detected by a non-validating processor. Examples include the constraints entitled Entity Declared, Parsed Entity, and No Recursion, as well as some of the cases described as forbidden in "4.4 XML Processor Treatment of Entities and References".
- The information passed from the processor to the application may vary, depending on whether the processor reads parameter and external entities. For example, a non-validating processor may not normalize attribute values, include the replacement text of internal entities, or supply default attribute values, where doing so depends on having read declarations in external or parameter entities.

For maximum reliability in interoperating between different XML processors, applications which use non-validating processors should not rely on any behaviors not required of such processors. Applications which require facilities such as the use of default attributes or internal entities which are declared in external entities should use validating XML processors.

6. Notation

The formal grammar of XML is given in this specification using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are defined by a regular expression, or with an initial lower case letter otherwise. Literal strings are quoted.

On utilisera, du côté droit d'une règle, les expressions suivantes qui correspondront à un ou plusieurs caractères :

`#xN`	où `N` est un entier hexadécimal. L'expression équivaut au caractère de l'ISO/CEI 10646 dont le point de code (UCS-4) a la valeur N, sans égard au nombre de zéros de tête de la forme `#xN` ; le nombre de zéros de ce point de code dépend en effet du codage utilisé.
`[a-zA-Z], [#xN-#xN]`	équivaut à n'importe quel caractère dont la valeur appartient à l'intervalle ouvert indiqué (inclusivement).
`[^a-z], [^#xN-#xN]`	équivaut à n'importe quel caractère dont la valeur n'appartient *pas* à l'intervalle indiqué.
`[^abc], [^#xN#xN#xN]`	équivaut à n'importe quel caractère autre que ceux mentionnés.
`"chaîne"`	équivaut à une chaîne littérale correspondant à celle indiquée entre guillemets anglais doubles.
`'chaîne'`	équivaut à une chaîne littérale correspondant à celle indiquée entre guillemets anglais simples.

Ces symboles peuvent s'assembler pour construire des expressions plus complexes comme suit, où `A` et `B` représentent des expressions simples :

`(expression)`	l'`expression` est traitée comme une unité et peut être agencée selon la description de la liste.
`A?`	équivaut à `A` ou à rien ; `A` facultatif.
`A B`	équivaut à `A` suivi de `B`.
`A | B`	équivaut à `A` ou à `B` mais pas aux deux.
`A - B`	équivaut à n'importe quelle chaîne qui correspond à `A` mais ne correspond pas à `B`.
`A+`	équivaut à une ou plusieurs apparitions de `A`.
`A*`	équivaut à zéro, une ou plusieurs apparitions de `A`.

On trouvera aussi les notations suivantes dans les productions :

`/* ... */`	commentaire.
`[ cf: ... ]`	contrainte de forme ; identifie par un nom une contrainte associée à une production et imposée aux documents bien formés.
`[ cv: ... ]`	contrainte de validité ; identifie par un nom une contrainte associée à une production et imposée aux documents valides.

Within the expression on the right-hand side of a rule, the following expressions are used to match strings of one or more characters:

`#xN`	where `N` is a hexadecimal integer, the expression matches the character in ISO/IEC 10646 whose canonical (UCS-4) code value, when interpreted as an unsigned binary number, has the value indicated. The number of leading zeros in the `#xN` form is insignificant; the number of leading zeros in the corresponding code value is governed by the character encoding in use and is not significant for XML.
`[a-zA-Z], [#xN-#xN]`	matches any character with a value in the range(s) indicated (inclusive).
`[^a-z], [^#xN-#xN]`	matches any character with a value *outside* the range indicated.
`[^abc], [^#xN#xN#xN]`	matches any character with a value not among the characters given.
`"string"`	matches a literal string matching that given inside the double quotes.
`'string'`	matches a literal string matching that given inside the single quotes.

These symbols may be combined to match more complex patterns as follows, where `A` and `B` represent simple expressions:

`(expression)`	`expression` is treated as a unit and may be combined as described in this list.
`A?`	matches `A` or nothing; optional `A`.
`A B`	matches `A` followed by `B`.
`A | B`	matches `A` or `B` but not both.
`A - B`	matches any string that matches `A` but does not match `B`.
`A+`	matches one or more occurrences of `A`.
`A*`	matches zero or more occurrences of `A`.

Other notations used in the productions are:

`/* ... */`	comment.
`[ wfc: ... ]`	well-formedness constraint; this identifies by name a constraint on well-formed documents associated with a production.
`[ vc: ... ]`	validity constraint; this identifies by name a constraint on valid documents associated with a production.

Annexes

A. Bibliographie

A.1 Bibliographie normative

IANA (Internet Assigned Numbers Authority) *Official Names for Character Sets*, éd. Keld Simonsen et al. (`ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets`).

IETF RFC 1766 IETF (Internet Engineering Task Force). *RFC 1766 : Tags for the Identification of Languages*, éd. H. Alvestrand. 1995.

ISO 639 (Organisation internationale de normalisation). *ISO 639:1988 (F). Codes pour la représentation des noms de langue.* [Genève] : Organisation internationale de normalisation, 1988.

ISO 3166 (Organisation internationale de normalisation). *ISO 3166-1:1997 (F). Codes pour la représentation des noms de pays et de leurs subdivisions – Partie 1: Codes pays.* [Genève] : Organisation internationale de normalisation, 1997.

ISO/CEI 10646 ISO (Organisation internationale de normalisation). *ISO/CEI 10646-1993 (F). Technologies de l'information – Jeu universel de caractères codés à plusieurs octets – Partie 1: Architecture et table multilingue.* [Genève] : Organisation internationale de normalisation, 1993 (ainsi que les amendements AM 1 à AM 7). Texte provisoire de la traduction française disponible à l'URL `http://babel.alis.com/codage/iso10646/index.html`.

Unicode The Unicode Consortium. *The Unicode Standard, version 2.0.* Reading, Massachusetts : Addison-Wesley Developers Press, 1996.

A.2 Autres ouvrages

Aho/Ullman Aho, Alfred V., Ravi Sethi et Jeffrey D. Ullman. *Compilateurs: Principes, techniques et outils.* InterÉditions, Paris, 1994.

Berners-Lee et al. Berners-Lee, T., R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax and Semantics.* 1997. (En cours de rédaction, voir les mises à jour au RFC1738.)

Brüggemann-Klein Brüggemann-Klein, Anne. *Regular Expressions into Finite Automata.* Résumé étoffé dans I. Simon, Hrsg., LATIN 1992, S. 97-98. Springer-Verlag, Berlin 1992. Version complète dans Theoretical Computer Science 120 : 197-213, 1993.

Brüggemann-Klein and Wood Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages.* Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991.

Clark James Clark. Comparison of SGML and XML. (`http://www.w3.org/TR/NOTE-sgml-xml-971215`).

IETF RFC1738 IETF (Internet Engineering Task Force). *RFC 1738 : Uniform Resource Locators (URL)*, éd. T. Berners-Lee, L. Masinter, M. McCahill. 1994.

IETF RFC1808 IETF (Internet Engineering Task Force). *RFC 1808 : Relative Uniform Resource Locators*, éd. R. Fielding. 1995.

Appendices

A. References

A.1 Normative References

IANA (Internet Assigned Numbers Authority) *Official Names for Character Sets*, ed. Keld Simonsen et al. See ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets (`ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets`).

IETF RFC 1766 IETF (Internet Engineering Task Force). *RFC 1766: Tags for the Identification of Languages*, ed. H. Alvestrand. 1995.

ISO 639 (International Organization for Standardization). *ISO 639:1988 (E). Code for the representation of names of languages.* [Geneva]: International Organization for Standardization, 1988.

ISO 3166 (International Organization for Standardization). *ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions – Part 1: Country codes* [Geneva]: International Organization for Standardization, 1997.

ISO/IEC 10646 ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

Unicode The Unicode Consortium. *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

A.2 Other References

Aho/Ullman Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Berners-Lee et al. Berners-Lee, T., R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax and Semantics*. 1997. (Work in progress; see updates to RFC1738.)

Brüggemann-Klein Brüggemann-Klein, Anne. *Regular Expressions into Finite Automata*. Extended abstract in I. Simon, Hrsg., LATIN 1992, S. 97-98. Springer-Verlag, Berlin 1992. Full Version in Theoretical Computer Science 120: 197-213, 1993.

Brüggemann-Klein and Wood Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991.

Clark James Clark. Comparison of SGML and XML. See http://www.w3.org/TR/NOTE-sgml-xml-971215 (`http://www.w3.org/TR/NOTE-sgml-xml-971215`).

IETF RFC1738 IETF (Internet Engineering Task Force). *RFC 1738: Uniform Resource Locators (URL)*, ed. T. Berners-Lee, L. Masinter, M. McCahill. 1994.

IETF RFC1808 IETF (Internet Engineering Task Force). *RFC 1808: Relative Uniform Resource Locators*, ed. R. Fielding. 1995.

IETF RFC2141 IETF (Internet Engineering Task Force). *RFC 2141 : URN Syntax*, éd. R. Moats. 1997.

ISO 8879 ISO (Organisation internationale de normalisation). *ISO 8879:1986 (F). Traitement de l'information – Systèmes bureautiques – Langage normalisé de balisage généralisé (SGML) (Incorpore l'amendement 1:1988).* [Genève] : Organisation internationale de normalisation, 1986.

ISO/IEC 10744 ISO (Organisation internationale de normalisation). *ISO/IEC 10744-1992 (E). Technologies de l'information – Langage de structuration temporelle/hypermédia (HyTime) (Publiée actuellement en anglais seulement)* [Genève] : Organisation internationale de normalisation, 1992. *Extended Facilities Annexe.* [Genève] : Organisation internationale de normalisation, 1996.

B. Classes de caractères

Conformément aux caractéristiques définies dans le standard Unicode, les caractères sont classés en caractères de base (parmi lesquels on retrouve les lettres de l'alphabet latin sans diacritiques), en caractères idéographiques et en caractères jonctifs (on y retrouve notamment la plupart des diacritiques). Ces classes forment la classe des lettres. On distingue aussi les chiffres et les modificateurs de lettre.

Les caractères

```
[84] Lettre   ::=  CarBase | Idéogramme
[85] CarBase  ::= [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6]
                | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131]
                | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E]
                | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5]
                | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1] | #x0386
                | [#x0388-#x038A] | #x038C | [#x038E-#x03A1] | [#x03A3-#x03CE]
                | [#x03D0-#x03D6] | #x03DA | #x03DC | #x03DE | #x03E0
                | [#x03E2-#x03F3] | [#x0401-#x040C] | [#x040E-#x044F]
                | [#x0451-#x045C] | [#x045E-#x0481] | [#x0490-#x04C4]
                | [#x04C7-#x04C8] | [#x04CB-#x04CC] | [#x04D0-#x04EB]
                | [#x04EE-#x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559
                | [#x0561-#x0586] | [#x05D0-#x05EA] | [#x05F0-#x05F2]
                | [#x0621-#x063A] | [#x0641-#x064A] | [#x0671-#x06B7]
                | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5
                | [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D | [#x0958-#x0961]
                | [#x0985-#x098C] | [#x098F-#x0990] | [#x0993-#x09A8]
                | [#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD]
                | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A]
                | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30]
                | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39]
                | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B]
                | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0]
                | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0
                | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28]
                | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D
                | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61] | [#x0B85-#x0B8A]
```

IETF RFC2141 IETF (Internet Engineering Task Force). *RFC 2141: URN Syntax*, ed. R. Moats. 1997.

ISO 8879 ISO (International Organization for Standardization). *ISO 8879:1986(E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML).* First edition – 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10744 ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology – Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996.

B. Character Classes

Following the characteristics defined in the Unicode standard, characters are classed as base characters (among others, these contain the alphabetic characters of the Latin alphabet, without diacritics), ideographic characters, and combining characters (among others, this class contains most diacritics); these classes combine to form the class of letters. Digits and extenders are also distinguished.

Characters

```
[84] Letter   ::= BaseChar | Ideographic

[85] BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6]
                | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131]
                | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E]
                | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5]
                | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1] | #x0386
                | [#x0388-#x038A] | #x038C | [#x038E-#x03A1] | [#x03A3-#x03CE]
                | [#x03D0-#x03D6] | #x03DA | #x03DC | #x03DE | #x03E0
                | [#x03E2-#x03F3] | [#x0401-#x040C] | [#x040E-#x044F]
                | [#x0451-#x045C] | [#x045E-#x0481] | [#x0490-#x04C4]
                | [#x04C7-#x04C8] | [#x04CB-#x04CC] | [#x04D0-#x04EB]
                | [#x04EE-#x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559
                | [#x0561-#x0586] | [#x05D0-#x05EA] | [#x05F0-#x05F2]
                | [#x0621-#x063A] | [#x0641-#x064A] | [#x0671-#x06B7]
                | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5
                | [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D | [#x0958-#x0961]
                | [#x0985-#x098C] | [#x098F-#x0990] | [#x0993-#x09A8]
                | [#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD]
                | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A]
                | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30]
                | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39]
                | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B]
                | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0]
                | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0
                | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28]
                | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D
                | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61] | [#x0B85-#x0B8A]
```

```
                  | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A] | #x0B9C
                  | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA]
                  | [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C]
                  | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33]
                  | [#x0C35-#x0C39] | [#x0C60-#x0C61] | [#x0C85-#x0C8C]
                  | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3]
                  | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C]
                  | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39]
                  | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33]
                  | [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88]
                  | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-#x0E9F]
                  | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB]
                  | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD
                  | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69]
                  | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103]
                  | [#x1105-#x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112]
                  | #x113C | #x113E | #x1140 | #x114C | #x114E | #x1150
                  | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163 | #x1165
                  | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175
                  | #x119E | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8]
                  | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9
                  | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15]
                  | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D]
                  | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-#x1F7D]
                  | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE | [#x1FC2-#x1FC4]
                  | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB]
                  | [#x1FE0-#x1FEC] | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126
                  | [#x212A-#x212B] | #x212E | [#x2180-#x2182] | [#x3041-#x3094]
                  | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-#xD7A3]

[86] Idéogramme ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

[87] CarJonctif ::= [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486]
                  | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD] | #x05BF
                  | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652] | #x0670
                  | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4]
                  | [#x06E7-#x06E8] | [#x06EA-#x06ED] | [#x0901-#x0903] | #x093C
                  | [#x093E-#x094C] | #x094D | [#x0951-#x0954] | [#x0962-#x0963]
                  | [#x0981-#x0983] | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4]
                  | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7 | [#x09E2-#x09E3]
                  | #x0A02 | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42]
                  | [#x0A47-#x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71]
                  | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9]
                  | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43]
                  | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57]
                  | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8]
                  | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44]
                  | [#x0C46-#x0C48] | [#x0C4A-#x0C4D] | [#x0C55-#x0C56]
                  | [#x0C82-#x0C83] | [#x0CBE-#x0CC4] | [#x0CC6-#x0CC8]
                  | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6] | [#x0D02-#x0D03]
                  | [#x0D3E-#x0D43] | [#x0D46-#x0D48] | [#x0D4A-#x0D4D] | #x0D57
                  | #x0E31 | [#x0E34-#x0E3A] | [#x0E47-#x0E4E] | #x0EB1
                  | [#x0EB4-#x0EB9] | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD]
```

```
                  | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A] | #x0B9C
                  | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA]
                  | [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C]
                  | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33]
                  | [#x0C35-#x0C39] | [#x0C60-#x0C61] | [#x0C85-#x0C8C]
                  | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3]
                  | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C]
                  | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39]
                  | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33]
                  | [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88]
                  | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-#x0E9F]
                  | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB]
                  | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD
                  | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69]
                  | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103]
                  | [#x1105-#x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112]
                  | #x113C | #x113E | #x1140 | #x114C | #x114E | #x1150
                  | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163 | #x1165
                  | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175
                  | #x119E | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8]
                  | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9
                  | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15]
                  | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D]
                  | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-#x1F7D]
                  | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE | [#x1FC2-#x1FC4]
                  | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB]
                  | [#x1FE0-#x1FEC] | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126
                  | [#x212A-#x212B] | #x212E | [#x2180-#x2182] | [#x3041-#x3094]
                  | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-#xD7A3]

[86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

[87] CombiningChar ::= [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486]
                  | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD] | #x05BF
                  | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652] | #x0670
                  | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4]
                  | [#x06E7-#x06E8] | [#x06EA-#x06ED] | [#x0901-#x0903] | #x093C
                  | [#x093E-#x094C] | #x094D | [#x0951-#x0954] | [#x0962-#x0963]
                  | [#x0981-#x0983] | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4]
                  | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7 | [#x09E2-#x09E3]
                  | #x0A02 | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42]
                  | [#x0A47-#x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71]
                  | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9]
                  | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43]
                  | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57]
                  | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8]
                  | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44]
                  | [#x0C46-#x0C48] | [#x0C4A-#x0C4D] | [#x0C55-#x0C56]
                  | [#x0C82-#x0C83] | [#x0CBE-#x0CC4] | [#x0CC6-#x0CC8]
                  | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6] | [#x0D02-#x0D03]
                  | [#x0D3E-#x0D43] | [#x0D46-#x0D48] | [#x0D4A-#x0D4D] | #x0D57
                  | #x0E31 | [#x0E34-#x0E3A] | [#x0E47-#x0E4E] | #x0EB1
                  | [#x0EB4-#x0EB9] | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD]
```

```
                  | [#x0F18-#x0F19] | #x0F35 | #x0F37 | #x0F39 | #x0F3E | #x0F3F
                  | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-#x0F95] | #x0F97
                  | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7] | #x0FB9 | [#x20D0-#x20DC]
                  | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A

[88] Chiffre    ::= [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9]
                  | [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F]
                  | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF]
                  | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F]
                  | [#x0E50-#x0E59] | [#x0ED0-#x0ED9] | [#x0F20-#x0F29]

[89] ModificateurLettre ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46
                  | #x0EC6 | #x3005 | [#x3031-#x3035]| [#x309D-#x309E]
                  | [#x30FC-#x30FE]
```

Les classes de caractères définies ci-dessus peuvent être dérivées de la base de données des caractères d'Unicode de la façon suivante :

- Les caractères initiaux de nom doivent appartenir à une des catégories suivantes : Ll, Lu, Lo, Lt, Nl.
- Les autres caractères constitutifs de nom doivent appartenir à une des catégories Mc, Me, Mn, Lm, ou Nd.
- Les caractères dans la zone de compatibilité (c-à-d ceux dont la valeur est supérieure à #xF900 et inférieure à #xFFFE) ne sont pas admis dans les noms XML.
- Les caractères qui connaissent une décomposition de compatibilité (c-à-d ceux dont le champ 5 dans la base de données contient une « balise de formatage de compatibilité », indiqué par un champ 5 commençant par un « < ») ne sont pas permis.
- Les caractères suivants sont considérés des caractères initiaux de nom, car le fichier de propriétés les classe parmi les caractères alphabétiques : [#x02BB-#x02C1], #x0559, #x06E5, #x06E6.
- Les caractères #x20DD-#x20E0 sont exclus (conformément à la section 5.14 d'Unicode).
- Le caractère #x00B7 est classé parmi les modificateurs de lettre, le fichier de propriétés le classant de la sorte.
- Le caractère #x0387 est considéré comme un caractère constitutif de nom car #x00B7 est son équivalent canonique.
- Les caractères « : » et « _ » sont permis comme caractères initiaux de nom.
- Les caractères « - » et « . » sont permis comme caractères constitutifs de nom.

C. XML et SGML (annexe informative)

XML a été conçu de telle sorte qu'il soit un sous-ensemble de SGML, ce qui signifie que chaque document XML valide devrait également être un document SGML conforme. Pour une comparaison détaillée des restrictions que XML impose aux documents au-delà de celles imposées par SGML, voir [Clark].

```
                  | [#x0F18-#x0F19] | #x0F35 | #x0F37 | #x0F39 | #x0F3E | #x0F3F
                  | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-#x0F95] | #x0F97
                  | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7] | #x0FB9 | [#x20D0-#x20DC]
                  | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A

[88] Digit      ::= [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9]
                  | [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F]
                  | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF]
                  | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F]
                  | [#x0E50-#x0E59] | [#x0ED0-#x0ED9] | [#x0F20-#x0F29]

[89] Extender   ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46
                  | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E]
                  | [#x30FC-#x30FE]}
```

The character classes defined here can be derived from the Unicode character database as follows:

- Name start characters must have one of the categories Ll, Lu, Lo, Lt, Nl.
- Name characters other than Name-start characters must have one of the categories Mc, Me, Mn, Lm, or Nd.
- Characters in the compatibility area (i.e. with character code greater than #xF900 and less than #xFFFE) are not allowed in XML names.
- Characters which have a font or compatibility decomposition (i.e. those with a "compatibility formatting tag" in field 5 of the database – marked by field 5 beginning with a "<") are not allowed.
- The following characters are treated as name-start characters rather than name characters, because the property file classifies them as Alphabetic: [#x02BB-#x02C1], #x0559, #x06E5, #x06E6.
- Characters #x20DD-#x20E0 are excluded (in accordance with Unicode, section 5.14).
- Character #x00B7 is classified as an extender, because the property list so identifies it.
- Character #x0387 is added as a name character, because #x00B7 is its canonical equivalent.
- Characters ':' and '_' are allowed as name-start characters.
- Characters '-' and '.' are allowed as name characters.

C. XML and SGML (Non-Normative)

XML is designed to be a subset of SGML, in that every valid XML document should also be a conformant SGML document. For a detailed comparison of the additional restrictions that XML places on documents beyond those of SGML, see [Clark].

D. Développement des appels d'entité et de caractères (annexe informative)

Cette annexe illustre par quelques exemples la suite des opérations de reconnaissance et de développement qui ont lieu lors du traitement des appels d'entité et de caractères, tel que décrit dans « 4.4 Traitement des entités et des appels par un processeur XML ».

Si la DTD contient la déclaration suivante :

```
<!ENTITY exemple "<p>On peut déguiser une
esperluette (&#38;#38;)
soit par un appel de caractère (&#38;#38;#38;)
soit par un appel d'entité générale
(&amp;amp;).</p>" >
```

alors le processeur XML reconnaîtra les appels de caractères lors de l'analyse de la déclaration d'entité et les résoudra avant de stocker la chaîne suivante comme valeur de l'entité « exemple » :

```
<p>On peut déguiser une esperluette (&#38;)
soit par un appel de caractère (&#38;#38;)
soit par un appel d'entité générale (&amp;amp;).</p>
```

Un appel à « `&exemple;` » au sein du document enclenchera une nouvelle analyse du texte, lors de laquelle on reconnaîtra les balises ouvrante et fermante de l'élément « p » ainsi que les trois appels qui seront développés. On aura donc comme résultat un élément « p » avec le contenu suivant (rien que des données, pas de délimiteur ni de balisage) :

```
On peut représenter une esperluette (&)
soit par un appel de caractère (&#38;)
soit par un appel d'entité générale (&amp;).
```

Un exemple plus complexe illustrera complètement les règles en question et leurs effets. Dans l'exemple ci-dessous, les numéros de ligne ne servent que de référence.

```
1 <?xml version='1.0'?>
2 <!DOCTYPE essai [
3 <!ELEMENT essai (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY risquée "sujet à
6 %xx;
7 ]>
8 <essai>Cet exemple illustre une méthode &risquée;.</essai>
```

Ce qui a pour résultat :

— à la ligne 4, de développer d'emblée l'appel au caractère 37, et de stocker l'entité paramètre « `xx` » dans la table des symboles avec comme valeur « `%zz;` ». Puisqu'on ne réanalyse pas le texte de remplacement, il n'y a pas reconnaissance de l'appel à l'entité paramètre « zz » par le processur (reconnaissance qui aurait causé une erreur, « zz » n'étant pas encore déclarée.)

— à la ligne 5, de développer sur le champ l'appel de caractère « `&#60;` » et de stocker l'entité paramètre « zz » avec comme texte de remplacement « `<!ENTITY risquée "sujet à erreurs" >` », une déclaration d'entité bien formée.

D. Expansion of Entity and Character References (Non-Normative)

This appendix contains some examples illustrating the sequence of entity- and character-reference recognition and expansion, as specified in "4.4 XML Processor Treatment of Entities and References".

If the DTD contains the declaration

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>" >
```

then the XML processor will recognize the character references when it parses the entity declaration, and resolve them before storing the following string as the value of the entity "example":

```
<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

A reference in the document to "&example;" will cause the text to be reparsed, at which time the start- and end-tags of the "p" element will be recognized and the three references will be recognized and expanded, resulting in a "p" element with the following content (all data, no delimiters or markup):

```
An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).
```

A more complex example will illustrate the rules and their effects fully. In the following example, the line numbers are solely for reference.

```
1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

This produces the following:

- in line 4, the reference to character 37 is expanded immediately, and the parameter entity "xx" is stored in the symbol table with the value "%zz;". Since the replacement text is not rescanned, the reference to parameter entity "zz" is not recognized. (And it would be an error if it were, since "zz" is not yet declared.)
- in line 5, the character reference "<" is expanded immediately and the parameter entity "zz" is stored with the replacement text "<!ENTITY tricky "error-prone" >", which is a well-formed entity declaration.

— à la ligne 6, de reconnaître l'appel à « xx » et d'analyser le texte de remplacement de « xx » (à savoir « %zz; »). Le processeur reconnaît alors l'appel à « zz » et analyse son texte de remplacement (« <!ENTITY risquée "sujet à erreurs" > »). L'entité générale « risquée » est donc maintenant déclarée avec comme texte de remplacement « sujet à erreurs ».
— à la ligne 8, de reconnaître l'appel à l'entité générale « risquée » et de la développer de telle sorte que le contenu de l'élément « essai » est la chaîne auto-descriptive (et agrammaticale) *Cet exemple illustre une méthode sujet à erreurs.*

E. Modèles de contenu déterministes (annexe informative)

Pour favoriser la compatibilité, il est nécessaire que les modèles de contenu des déclarations de type d'éléments soient déterministes.

SGML exige des modèles de contenu déterministes (appelés « non-ambiguës ») ; les processeurs XML construits sur la base de systèmes SGML peuvent signaler comme erreurs les modèles de contenu non-déterministes.

Par exemple, le modèle de contenu `((b, c) | (b, d))` est non-déterministe, car étant donné un `b` initial, l'analyseur ne peut déterminer à quel `b` du modèle il correspond sans savoir d'avance quel élément suit le `b`. Dans ce cas-ci, les deux références à`b` peuvent être fusionnés en une seule référence, le modèle devenant `(b, (c | d))`. Un `b` initial correspond maintenant clairement à un seul nom dans le modèle de contenu. L'analyseur n'a plus besoin de prévoir ce qui suit ; `c` ou `d` serait accepté.

Formellement : un automate à états finis peut être construit, à partir du modèle de contenu, en utilisant les algorithmes standards comme l'algorithme 3.5 dans la section 3.9 de Aho, Sethi et Ullman [Aho/Ullman]. Dans plusieurs de ces algorithmes, un ensemble des suivants est construit pour chaque position dans l'expression régulière (c-à-d chaque feuille dans l'arbre syntaxique de l'expression régulière) ; si une des positions a un ensemble des suivants dans lequel plus d'une des positions suivantes a le même nom de type d'élément, alors le modèle de contenu est erroné et peut être signalé comme tel.

Il existe des algorithmes permettant de réduire automatiquement beaucoup de modèles de contenu non-déterministes (mais pas tous) à des modèles déterministes équivalents ; cf. Brüggemann-Klein 1991 [Brüggemann-Klein].

F. Auto-détection du codage de caractères (annexe informative)

La déclaration de codage XML tient le rôle d'étiquette interne à chaque entité, indiquant quel codage de caractères y est utilisé. Pour qu'un processeur XML puisse lire l'étiquette interne, toutefois, il semble devoir connaître le codage utilisé, ce qui est précisément ce que l'étiquette cherche à annoncer. En général, la situation est sans issue. Le cas est néanmoins meilleur en XML, puisque XML limite le cas général de deux façons : un processeur donné ne gère qu'un nombre fini de codages de caractères ; il y a des restrictions sur la position et le contenu de la déclaration de codage XML, restrictions telles qu'il est faisable de détecter automatiquement le codage de chaque entité dans les cas normaux. De plus, dans bien des cas d'autres sources d'information sont disponibles en sus des données XML elles-mêmes. Deux cas sont à distinguer, selon qu'une entité XML est présentée au processeur avec ou sans information externe sur le codage. Nous considérons d'abord le dernier cas.

- in line 6, the reference to "`xx`" is recognized, and the replacement text of "`xx`" (namely "`%zz;`") is parsed. The reference to "`zz`" is recognized in its turn, and its replacement text ("`<!ENTITY tricky "error-prone" >`") is parsed. The general entity "`tricky`" has now been declared, with the replacement text "`error-prone`".
- in line 8, the reference to the general entity "`tricky`" is recognized, and it is expanded, so the full content of the "`test`" element is the self-describing (and ungrammatical) string *This sample shows a error-prone method.*

E. Deterministic Content Models (Non-Normative)

For compatibility, it is required that content models in element type declarations be deterministic.

SGML requires deterministic content models (it calls them "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.

For example, the content model `((b, c) | (b, d))` is non-deterministic, because given an initial `b` the parser cannot know which `b` in the model is being matched without looking ahead to see which element follows the `b`. In this case, the two references to `b` can be collapsed into a single reference, making the model read `(b, (c | d))`. An initial `b` now clearly matches only a single name in the content model. The parser doesn't need to look ahead to see what follows; either `c` or `d` would be accepted.

More formally: a finite state automaton may be constructed from the content model using the standard algorithms, e.g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman [Aho/Ullman]. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error.

Algorithms exist which allow many but not all non-deterministic content models to be reduced automatically to equivalent deterministic models; see Brüggemann-Klein 1991 [Brüggemann-Klein].

F. Autodetection of Character Encodings (Non-Normative)

The XML encoding declaration functions as an internal label on each entity, indicating which character encoding is in use. Before an XML processor can read the internal label, however, it apparently has to know what character encoding is in use–which is what the internal label is trying to indicate. In the general case, this is a hopeless situation. It is not entirely hopeless in XML, however, because XML limits the general case in two ways: each implementation is assumed to support only a finite set of character encodings, and the XML encoding declaration is restricted in position and content in order to make it feasible to autodetect the character encoding in use in each entity in normal cases. Also, in many cases other sources of information are available in addition to the XML data stream itself. Two cases may be distinguished, depending on whether the XML entity is presented to the processor without, or with, any accompanying (external) information. We consider the first case first.

Étant donné que toute entité XML codée autrement qu'en UTF-8 ou UTF-16 *doit* commencer par une déclaration de codage, dont les premiers caractères doivent être '`<?xml`', tout processeur conforme peut déterminer, après avoir lu de deux à quatre octets, lequel des cas suivants s'applique. En lisant cette liste, il peut être utile de se rappeler qu'en UCS-4, '<' et '?' sont représentés par "`#x0000003C`" et"`#x0000003F`" respectivement, alors que la marque d'ordre d'octets exigée en UTF-16 est "`#xFEFF`".

— `00 00 00 3C`: UCS-4, machine grand-boutienne (ordre 1234)
— `3C 00 00 00`: UCS-4, machine petit-boutienne (ordre 4321)
— `00 00 3C 00`: UCS-4, ordre des octets inhabituel (2143)
— `00 3C 00 00`: UCS-4, ordre des octets inhabituel (3412)
— `FE FF`: UTF-16, grand-boutien
— `FF FE`: UTF-16, petit-boutien
— `00 3C 00 3F`: UTF-16, grand-boutien, pas de marque d'ordre d'octets (et donc, strictement, une erreur)
— `3C 00 3F 00`: UTF-16, petit-boutien, pas de marque d'ordre d'octets (et donc, strictement, une erreur)
— `3C 3F 78 6D`: UTF-8, ISO 646, ASCII, quelconque partie de l'ISO 8859, Shift-JIS, EUC, ou tout autre codage à 7 bits, 8 bits ou de largeur variable tel que les caractères ASCII ont leur position, largeur et valeur habituelle ; la déclaration de codage doit être lue pour déterminer exactement de quel codage il s'agit, ce qui peut être fait de façon fiable puisque tous ces codages utilisent les mêmes bits pour les caractères ASCII.
— `4C 6F A7 94`: EBCDIC (quelconque ; la déclaration de codage doit être lue pour déterminer la page de code exacte).
— autre: UTF-8 sans déclaration de codage, ou alors le flux de données est corrompu, fragmentaire ou emballé de quelque façon.

Ce niveau d'auto-détection est suffisant pour lire la déclaration de codage XML et en tirer le nom du codage, qui est nécessaire pour distinguer les membres de chaque famille de codage (par ex. distinguer UTF-8 de 8859, distinguer les parties de 8859, distinguer les diverses pages de code EBCDIC, etc.)

Puisque le contenu de la déclaration de codage est limité aux caractères ASCII, un processeur peut la lire de façon fiable dès qu'il a identifié la famille de codage utilisée. En pratique, tous les codages de caractères font partie d'une des familles sus-mentionnées ; la déclaration de codage permet donc une identification raisonnablement robuste du codage des caractères, même en l'absence d'une source fiable d'information externe (système d'exploitation ou protocole de transport).

Dès que le processeur a déterminé le codage utilisé, il peut agir en conséquence, en appelant une routine d'entrée pour chaque cas ou en appelant la fonction de conversion appropriée pour chaque caractère.

Comme tout système d'auto-étiquetage, la déclaration de codage XML cesse de fonctionner si un logiciel change le codage de l'entité sans la mettre à jour. Les concepteurs de routines de codage de caractères doivent être attentifs pour assurer l'exactitude de l'information interne ou externe rattachée à l'entité.

Because each XML entity not in UTF-8 or UTF-16 format *must* begin with an XML encoding declaration, in which the first characters must be '<?xml', any conforming processor can detect, after two to four octets of input, which of the following cases apply. In reading this list, it may help to know that in UCS-4, '<' is "#x0000003C" and '?' is "#x0000003F", and the Byte Order Mark required of UTF-16 data streams is "#xFEFF".

- 00 00 00 3C: UCS-4, big-endian machine (1234 order)
- 3C 00 00 00: UCS-4, little-endian machine (4321 order)
- 00 00 3C 00: UCS-4, unusual octet order (2143)
- 00 3C 00 00: UCS-4, unusual octet order (3412)
- FE FF: UTF-16, big-endian
- FF FE: UTF-16, little-endian
- 00 3C 00 3F: UTF-16, big-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- 3C 00 3F 00: UTF-16, little-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- 3C 3F 78 6D: UTF-8, ISO 646, ASCII, some part of ISO 8859, Shift-JIS, EUC, or any other 7-bit, 8-bit, or mixed-width encoding which ensures that the characters of ASCII have their normal positions, width, and values; the actual encoding declaration must be read to detect which of these applies, but since all of these encodings use the same bit patterns for the ASCII characters, the encoding declaration itself may be read reliably
- 4C 6F A7 94: EBCDIC (in some flavor; the full encoding declaration must be read to tell which code page is in use)
- other: UTF-8 without an encoding declaration, or else the data stream is corrupt, fragmentary, or enclosed in a wrapper of some kind

This level of autodetection is enough to read the XML encoding declaration and parse the character-encoding identifier, which is still necessary to distinguish the individual members of each family of encodings (e.g. to tell UTF-8 from 8859, and the parts of 8859 from each other, or to distinguish the specific EBCDIC code page in use, and so on).

Because the contents of the encoding declaration are restricted to ASCII characters, a processor can reliably read the entire encoding declaration as soon as it has detected which family of encodings is in use. Since in practice, all widely used character encodings fall into one of the categories above, the XML encoding declaration allows reasonably reliable in-band labeling of character encodings, even when external sources of information at the operating-system or transport-protocol level are unreliable.

Once the processor has detected the character encoding in use, it can act appropriately, whether by invoking a separate input routine for each case, or by calling the proper conversion function on each character of input.

Like any self-labeling system, the XML encoding declaration will not work if any software changes the entity's character set or encoding without updating the encoding declaration. Implementors of character-encoding routines should be careful to ensure the accuracy of the internal and external information used to label the entity.

Le second cas de figure se présente lorsque l'entité XML est accompagnée d'information sur le codage, fournie par certains système de fichier et certains protocoles réseau. En présence de plusieurs sources d'information, il convient que le protocole de haut niveau qui transporte l'entité précise leur priorités relatives ainsi que la méthode préconisée pour résoudre les conflits potentiels. Des règles de priorité relative de l'étiquette interne et de l'étiquette de type MIME d'un en-tête externe, par exemple, devrait être précisées dans le document RFC définissant les types MIME text/xml et application/xml. On recommande toutefois les règles suivantes, dans l'intérêt de l'interopérabilité :

— Si une entité XML se trouve dans un fichier, la marque d'ordre d'octet et la déclaration de codage (le cas échéant) sont utilisées pour déterminer le codage de caractères. Toute autre source d'information ne doit servir qu'à la récupération d'erreur.
— Si une entité XML est transportée avec le type MIME text/xml, le paramètre `charset` détermine le codage des caractères. Toute autre source d'information ne doit servir qu'à la récupération d'erreur.
— Si une entité XML est transportée avec le type MIME application/xml, la marque d'ordre d'octet et la déclaration de codage (le cas échéant) sont utilisées pour déterminer le codage de caractères. Toute autre source d'information ne doit servir qu'à la récupération d'erreur.

Ces règles ne s'appliquent que lorsque le niveau du protocole n'en précise pas d'autres ; en particulier, dès que les types MIME text/xml et application/xml seront définis, les recommendations du RFC pertinent s'appliqueront.

G. Groupe de travail XML du W3C (annexe informative)

Cette recommendation a été préparée et approuvée pour publication par le groupe de travail XML du W2C (le GT). L'approbation du GT ne signifie pas que tous les membres du GT ont voté positivement. Les membres actuels et passés du GT sont :

Jon Bosak, Sun (président) ; James Clark (meneur technique) ; Tim Bray, Textuality et Netscape (co-éditeur XML) ; Jean Paoli, Microsoft (co-éditeur XML) ; C. M. Sperberg-McQueen, U. d'Ill. (co-éditeur XML) ; Dan Connolly, W3C (liaison W3C) ; Paula Angerstein, Texcel ; Steve DeRose, INSO ; Dave Hollander, HP ; Eliot Kimber, ISOGEN ; Eve Maler, ArborText ; Tom Magliery, NCSA ; Murray Maloney, Muzmo et Grif ; Makoto Murata, Fuji Xerox Information Systems ; Joel Nava, Adobe ; Conleth O'Connell, Vignette ; Peter Sharpe, SoftQuad ; John Tigue, DataChannel.

The second possible case occurs when the XML entity is accompanied by encoding information, as in some file systems and some network protocols. When multiple sources of information are available, their relative priority and the preferred method of handling conflict should be specified as part of the higher-level protocol used to deliver XML. Rules for the relative priority of the internal label and the MIME-type label in an external header, for example, should be part of the RFC document defining the text/xml and application/xml MIME types. In the interests of interoperability, however, the following rules are recommended.

- If an XML entity is in a file, the Byte-Order Mark and encoding-declaration PI are used (if present) to determine the character encoding. All other heuristics and sources of information are solely for error recovery.
- If an XML entity is delivered with a MIME type of text/xml, then the `charset` parameter on the MIME type determines the character encoding method; all other heuristics and sources of information are solely for error recovery.
- If an XML entity is delivered with a MIME type of application/xml, then the Byte-Order Mark and encoding-declaration PI are used (if present) to determine the character encoding. All other heuristics and sources of information are solely for error recovery.

These rules apply only in the absence of protocol-level documentation; in particular, when the MIME types text/xml and application/xml are defined, the recommendations of the relevant RFC will supersede these rules.

G. W3C XML Working Group (Non-Normative)

This specification was prepared and approved for publication by the W3C XML Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval. The current and former members of the XML WG are:

Jon Bosak, Sun (Chair); James Clark (Technical Lead); Tim Bray, Textuality and Netscape (XML Co-editor); Jean Paoli, Microsoft (XML Co-editor); C. M. Sperberg-McQueen, U. of Ill. (XML Co-editor); Dan Connolly, W3C (W3C Liaison); Paula Angerstein, Texcel; Steve DeRose, INSO; Dave Hollander, HP; Eliot Kimber, ISOGEN; Eve Maler, ArborText; Tom Magliery, NCSA; Murray Maloney, Muzmo and Grif; Makoto Murata, Fuji Xerox Information Systems; Joel Nava, Adobe; Conleth O'Connell, Vignette; Peter Sharpe, SoftQuad; John Tigue, DataChannel

Notes de traduction

Cette traduction a été effectuée sur une période de plusieurs mois débutant en décembre 1998. Elle s'est déroulé en trois étapes principales : terminologie, traduction et révision. Pour faciliter le travail, un espace Web et une liste de diffusion ont été créés.

Terminologie

La première étape a été d'identifier la terminologie spécialisée à traduire et les traductions appropriées. En plus des traducteurs, les personnes suivantes ont participé à l'étude de terminologie (`file:///D|/Docs/HTML/Babel/web_ml/xml/termino.html`) : Alain LaBonté (courriel : `<alb@sct.gouv.qc.ca>`), Nicolas Lesbats (courriel : `<nlesbats@hotmail.com>`) et Philippe Deschamp (courriel : `<Philippe.Deschamp@INRIA.Fr>`).

Traduction

Le document original a été divisé en sections (matière liminaire, chapitres et annexes) que se sont partagés les traducteurs. Une fois les traductions faites, le document traduit fut réassemblé au moyen d'un script perl ad hoc qui s'est révélé fort utile pendant l'étape de révision.

Révision

Dans un premier temps, chaque traducteur a révisé certaines des sections traduites par les autres traducteurs ; toutes les sections ont ainsi été révisées au moins une fois. Cette étape a conduit à un affinement de la terminologie, à son uniformisation et à la correction de nombreuses erreurs. Quelques erreurs de l'original ont aussi été découvertes, qui ont été soumises au groupe de travail idoine du W3C.

Le document entier a ensuite été relu attentivement par Alain LaBonté (courriel : `<alb@sct.gouv.qc.ca>`) et Richard Parent (courriel : `<rparent@sct.gouv.qc.ca>`) qui ont relevé quelques erreurs et suggéré quelques améliorations à la terminologie. L'ébauche étant en permanence sur le Web et une certaine publicité ayant été faite de sa présence, quelques personnes nous ont contacté pour signaler des erreurs ou améliorations souhaitables.

Finalement, le document traduit et révisé plusieurs fois a été validé au moyen du validateur HTML du W3C (`http://validator.w3.org/`).