国 际 电 信 联 盟

ITU-T X.891

国际电信联盟
电信标准化部门

(05/2005)

X系列：数据网、开放系统通信和安全性

OSI 应用 — ASN.1的一般应用

信息技术 — ASN.1的一般应用：快速信息集

ITU-T X.891建议书

国际电信联盟

ITU-T X系列建议书

数据网、开放系统通信和安全性

公众数据网	
业务和设施	X.1-X.19
接口	X.20-X.49
传输、信令和交换	X.50-X.89
网络概貌	X.90-X.149
维护	X.150-X.179
管理安排	X.180-X.199
开放系统互连	
模型和记法	X.200-X.209
服务限定	X.210-X.219
连接式协议规范	X.220-X.229
无连接式协议规范	X.230-X.239
PICS书写形式	X.240-X.259
协议标识	X.260-X.269
安全协议	X.270-X.279
层管理对象	X.280-X.289
一致性测试	X.290-X.299
网间互通	
概述	X.300-X.349
卫星数据传输系统	X.350-X.369
以IP为基础的网络	X.370-X.379
报文处理系统	X.400-X.499
号码簿	X.500-X.599
OSI 组网和系统概貌	
组网	X.600-X.629
效率	X.630-X.639
业务质量	X.640-X.649
命名、寻址和登记	X.650-X.679
抽象句法记法1(ASN.1)	X.680-X.699
OSI 管理	
系统管理框架和结构	X.700-X.709
管理通信服务和协议	X.710-X.719
管理信息的结构	X.720-X.729
管理功能	X.730-X.799
安全	X.800-X.849
OSI 应用	
托付、并发和恢复	X.850-X.859
事务处理	X.860-X.879
远程操作	X.880-X.889
ASN.1的一般应用	**X.890-X.899**
开放分布式处理	X.900-X.999
电信安全	X.1000-

欲了解更详细信息，请查阅*ITU-T*建议书目录。

国际标准 ISO/IEC 24824-1
ITU-T X.891建议书

信息技术 — ASN.1的一般应用：快速信息集

摘　要

本建议书 | 国际标准规定了 W3C XML 信息集的某个实例使用二进制编码的一种表示法。这种二进制编码使用 ASN.1 表示法和 ASN.1 编码控制表示法（ECN）来规定。

本建议书 | 国际标准中规定的技术被称为快速信息集。它为 W3C XML 语法提供了一种可替换方式，来表示 W3C XML 信息集的实例。一般来说，这种表示法比 W3C XML 表示法提供了更小的编码长度和更快的处理。

本建议书 | 国际标准规定了多种技术的用法，来使得编码长度最小化（称为快速信息集文档），并使得创建和处理快速信息集文档的速度最大化。这些技术包括使用动态表（为字符串和限定名字），初始词汇，以及外部词汇等。

本建议书 | 国际标准还规定了一种多用途互联网邮件扩展（MIME）媒体类型来标识一个快速信息集文档。

来　源

ITU-T 第 17 研究组（2005-2008）按照 ITU-T A.8 建议书规定的程序，于 2005 年 5 月 14 日批准了 ITU-T X.891 建议书。它包括 ITU-T 第 17 研究组（2005-2008）按照 ITU-T A.8 建议书规定的程序，于 2006 年 6 月 13 日批准的技术勘误 1 所引入的修正。相同的文本也发表为 ISO/IEC 24824-1。

前　言

国际电信联盟（ITU）是从事电信领域工作的联合国专门机构。ITU-T（国际电信联盟电信标准化部门）是国际电信联盟的常设机构，负责研究技术、操作和资费问题，并且为在世界范围内实现电信标准化，发表有关上述研究项目的建议书。

每四年一届的世界电信标准化全会（WTSA）确定 ITU-T 各研究组的研究课题，再由各研究组制定有关这些课题的建议书。

WTSA 第 1 号决议规定了批准建议书须遵循的程序。

属 ITU-T 研究范围的某些信息技术领域的必要标准，是与国际标准化组织（ISO）和国际电工技术委员会（IEC）合作制定的。

注

本建议书为简明扼要起见而使用的“主管部门”一词，既指电信主管部门，又指经认可的运营机构。

遵守本建议书的规定是以自愿为基础的，但建议书可能包含某些强制性条款（以确保例如互操作性或适用性等），只有满足所有强制性条款的规定，才能达到遵守建议书的目的。“应该”或“必须”等其它一些强制性用语及其否定形式被用于表达特定要求。使用此类用语不表示要求任何一方遵守本建议书。

知识产权

国际电联提请注意：本建议书的应用或实施可能涉及使用已申报的知识产权。国际电联对无论是其成员还是建议书制定程序之外的其它机构提出的有关已申报的知识产权的证据、有效性或适用性不表示意见。

至本建议书批准之日止，国际电联已收到实施本建议书可能需要的受专利保护的知识产权的通知。但需要提醒实施者注意的是，这可能并非最新信息，因此特大力提倡他们通过下列网址查询电信标准化局（TSB）的专利数据库：http://www.itu.int/ITU-T/ipr/。

目 录

引言

本建议书 | 国际标准规定了 W3C XML 信息集的某个实例使用二进制编码的一种表示法（使用 ASN.1 表示法和 ASN.1 编码控制表示法来规定）。本建议书 | 国际标准的本版本所规定的编码被标识为版本 1（见 12.9 节）。

本建议书 | 国际标准中规定的技术被命名为快速信息集。它为 W3C XML 语法提供了一种可替换方式，来表示 W3C XML 信息集的实例。一般来说，这种表示法比 W3C XML 表示法提供了更小的编码长度和更快的处理。

本建议书 | 国际标准中规定的 W3C XML 信息集的某个实例的表示被称为一个快速信息集文档。每个快速信息集文档是表示 W3C XML 信息集实例的某个 ASN.1 数据类型（**Document** 类型 — 见 7.2 节）的抽象值的编码。

本建议书 | 国际标准规定了多种技术的用法，来使得一个快速信息集文档的编码长度最小化，并使得创建和处理此文档的速度最大化。

这些技术都是基于词汇表的使用，允许使用典型的小的数字值（词汇表索引）来替换字符串，这些字符串（例如）在 W3C XML 信息集的某个实例的 XML 1.0 序列中构成元素或属性的名字。

定义了较大量的词汇表（见第 8 节），在这些词汇表中，最基本的（八种字符串表）是将典型的小的数字值映射为字符串。然而，还有其他的词汇表（元素名字表和属性名字表）提供了更深层次的间接性，将一个词汇表索引映射为三个词汇表索引的集合，分别标识一个前缀、一个命名空间名字和一个本地名字。

另一个重要的技术是使用一种有限字母词汇表。它所包含的条目是 ISO/IEC 10646 字符的一个子集。如果一个字符串需要被编码，而在此表中有一个条目对应该字符串，则对此字符串的编码可以通过这样的方式进行：标识此词汇表正在被使用，并给定词汇表索引，然后将每个字符进行编码，放在此特定的 ISO/IEC 10646 字符子集所需的最小数量的比特中。定义了一定数量的内置有限字母，这些有限字母总是构成此表的前几个条目，并且涵盖了通常出现的字符串，如日期和时间，数字值等。

一个更进一步的重要优化是使用编码算法词汇表。这种表标识了专门的编码，可以应用于通常出现的串，同样的，也是使用了一定数量的内置算法。例如，如果有一个字符串看起来象是某个整数的十进制表示，其范围为 –32768 到 32767，则该字符串的编码可以通过这样的方式进行：标识此词汇表正在被使用，并给定词汇表索引，然后将此数字编码为一个两个八位组的有符号整数。浮点数和这些数字的数组也可以以同样的方法来支持。

为了确保快速处理而不牺牲紧凑性，一个快速信息集文档中的许多组件（例如字符串和表示 XML 信息集中信息项的组件等）都是需要八位组对齐的，而其他组件（例如长度和词汇表索引）不必要八位组对齐，但总是以一个八位组的最后一个比特来结束的。为了为这些优化的编码提供一个正式的规范，使用了 ASN.1 编码控制表示法（在 ITU-T X.692 建议书 | ISO/IEC 8825-3 中定义）（见 A.2），但在实现时使用 ECN 工具并不是必须的，并且对编码提供了一个完整的描述（见附件 C）。

某个特定的快速信息集文档的词汇表可以根据文档头部的信息来初始化，并且一般来说可以被动态地加入，以便为编码者提供灵活性。初始词汇表可以通过一个引用来提供，引用某个其他已标识的快速信息集文档的最终词汇表集合（也可以通过其他方法）。因此，这种词汇引用可以通过更进一步地增加表来进行补充，以便为本文档提供初始词汇表。更进一步地向表中动态增加一般是在创建文档或处理文档的过程中完成的。

最后，还有一个机制可以提供给快速信息集文档的创建者，让其可以包含与快速信息集文档的可选附加处理相关的数据（被称为附加处理数据），并与一个 URI 一起使用，此 URI 标识了对附加处理数据的格式和语义的完整规范。如果 URI 不可知，或者它所规定的处理不被支持或不被要求，则此可选的附加处理数据将被快速信息文档的后续处理者所忽略。

注 — 这种附加处理数据的一个示例是这样一种数据，它提供索引以便可以对快速信息集文档的一部分进行即时访问，因此如果仅对快速信息集文档中的符合某个特定 XML 标签的一部分内容感兴趣的话，就不必对整个文档进行处理了。

附件 A 构成了本建议书 | 国际标准的完整组成部分，它包含一个 ASN.1 模块（见 ITU-T X.680 建议书 | ISO/IEC 8824-1）和两个 ECN 模块（EDM 和 ELM – 见 ITU-T X.692 建议书 | ISO/IEC 8825-3），这几个模块共同规定了 **Document** 类型的某个值的抽象内容和比特级编码，而该 **Document** 类型的值携带的是 W3C XML 信息集的某个实例的值。

附件 B 是本建议书 | 国际标准的组成部分，它包含了标识一个快速信息集文档的 MIME 媒体类型的规范。

附件 C 不是本建议书 | 国际标准的组成部分，它为在第 12 节和附件 A.2 中正式规定的编码提供了一种完整的描述。

附件 D 不是本建议书 | 国际标准的组成部分，它提供了由一些 XML 文档所生成的快速信息集文档的示例。附件 D 还给出了 XML 表示的长度和这些示例的快速信息集表示的长度。

国际标准
ITU-T 建议书

信息技术 — ASN.1的一般应用：快速信息集

1 范围

本建议书 | 国际标准规定了一种 ASN.1 类型（见 ITU-T X.680 建议书 | ISO/IEC 8824-1），其抽象值表示了 W3C XML 信息集的实例。还规定了使用 ASN.1 编码控制表示法（见 ITU-T X.692 建议书 | ISO/IEC 8825-3）来对这些值进行二进制编码。

注 — 这些编码被称为快速信息集文档。

本建议书 | 国际标准还规定了相关技术，可以：

— 使得快速信息集文档的长度最小化；

— 使得创建和处理快速信息集文档的速度最大化；

— 允许（快速信息集文档的创建者）规范附加处理数据。

前两种技术包含了对概念词汇表的使用。词汇表的集合及其条目的特性在本建议书 | 国际标准中给出了全面的定义，但它们在计算机内存中的表示不在本建议书 | 国际标准的定义范围之内。被当作外部词汇使用的词汇表的传送或存储规定、对外显示或规范的正式表示法等，也都不在本建议书 | 国际标准的定义范围之内。

第三种技术包含了对附加处理数据的规定，以及对一个标识此数据格式和语义的 URI 的规定。附加处理数据的特定格式的规范及其用法不在本建议书 | 国际标准的定义范围之内。

URI 可被用于标识最终的词汇，这些词汇可被用作某个新的初始词汇的一部分或全部，但是如何为特定的最终词汇分配特定的 URI 不在本建议书 | 国际标准的定义范围之内。

本建议书 | 国际标准规定了内置的有限字母表，如何通过枚举将更多的有限字母加入到词汇表中，以及如何使用这些词汇表来更有效地对字符串进行编码等。

本建议书 | 国际标准进一步规定了为某些字符串进行最优编码所需的内置编码算法，以及如何将 URI 所标识的更多的编码算法加入到词汇表中等，但是这些更多的编码算法的定义及其它们相关联的 URI 不在本建议书 | 国际标准的定义范围之内。

另外，本建议书 | 国际标准还规定了一种多用途互联网邮件扩展（MIME）媒体类型来标识一个快速信息集文档。

2 规范性参考文献

下列 ITU-T 建议书和其他参考文献的条款，在本建议书中的引用而构成本建议书| 国际标准的条款。在出版时，所指出的版本是有效的。所有的建议书和其它参考文献均会得到修订，本建议书| 国际标准的使用者应查证是否有可能使用下列建议书或其它参考文献的最新版本。国际电联电信标准化局有目前有效的 ITU-T 建议书的清单。IEC 和 ISO 的各成员有目前有效的国际标准的目录。IETF 有 RFC 的清单，以及被后续 RFC 所废弃的 RFC 清单。W3C 有当前有效的 W3C 建议书清单。本建议书引用的文件自成一体时不具备建议书的地位。

2.1 等同的建议书 | 国际标准

- ITU-T Recommendation X.667 (2004) | ISO/IEC 9834-8:2005, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components.*
- ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- ITU-T Recommendation X.681 (2002) | ISO/IEC 8824-2:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*. †
- ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*. †
- ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.* †
- ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER).* †
- ITU-T Recommendation X.691 (2002) | ISO/IEC 8825-2:2002, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*. †
- ITU-T Recommendation X.692 (2002) | ISO/IEC 8825-3:2002, *Information technology – ASN.1 encoding rules: Specification of Encoding Control Notation (ECN).*
- ITU-T Recommendation X.693 (2001) | ISO/IEC 8825-4:2002, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)*. †

注 — 关于ASN.1建议书 | 国际标准的完整集合如上所列，因为它们在本建议书 | 国际标准的特定使用中都是适用的。如果没有在本建议书 | 国际标准的文本中被直接引用，则有一个† 符号会加在参考文献上。

2.2 其他参考文献

- ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times.*
- ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS).*
- *The Unicode Standard, Version 4.0,* The Unicode Consortium *(Reading, MA, Addison-Wesley).*

  注 1 — 由 Unicode 所定义的图形字符（及其编码）与 ISO/IEC 10646-1 所定义的字符是相同的，但是将 Unicode 作为一个参考文献包含进来是因为它还规定了控制字符的名字，并且定义了缩略语 UTF-16BE。
- W3C XML 1.0:2004, *Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2000/REC-xml-20040204/.*
- W3C XML 1.1:2004, *Extensible Markup Language (XML) 1.1, W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2000/REC-xml11-20040204/.*

  注 2 — W3C XML 1.0 和 W3C XML 1.1 都作为参考文献被包含进来，是因为任何一个都不是另一个的子集。这些参考文献在 3.4.10 中被单独使用。
- W3C XML Information Set:2004, *XML Information Set (Second Edition), W3C Recommendation, Copyright © [04 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.*
- W3C XML Namespaces 1.0:1999, *Namespaces in XML, W3C Recommendation, Copyright © [14 January 1999] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/1999/REC-xm-lnames-19990114/.*
- W3C XML Namespaces 1.1:2004, *Namespaces in XML 1.1, W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2004/REC-xm-l-names11-20040204/.*

注 3 – W3C XML 命名空间 1.0 和 W3C XML 命名空间 1.1 都作为参考文献被包含进来，是因为任何一个都不是另一个的子集。这些参考文献在 3.4.10 中被单独使用。

- IETF RFC 2045 (1996), *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.*
- IETF RFC 2396 (1998), *Uniform Resource Identifiers (URI): Generic Syntax.*
- IEEE 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic.*

3 定义

本建议书 | 国际标准采用下列定义。

3.1 ASN.1术语

本建议书 | 国际标准采用 ITU-T X.680 建议书 | ISO/IEC 8824-1 中规定的下列术语：

a） 选择类型；

b） 序列类型；

c） sequence-of 类型。

3.2 ECN术语

本建议书 | 国际标准采用在 ITU-T X.692 建议书 | ISO/IEC 8825-3 中规定的下列术语：

a） 编码定义模块（EDM）；

b） 编码链接模块（ELM）。

3.3 ISO/IEC 10646术语

本建议书 | 国际标准采用在 ISO/IEC 10646 中规定的下列术语：

a） 基本多语言平面。

3.4 其他定义

3.4.1 Base64 Base64：一种编码机制，使用一种有限的 65 个字符的字母表（见 10.3 节和 IETF RFC 2045）来将一个字符串表示为一个八位组串值。

3.4.2 character string 字符串：ISO/IEC 10646 所定义的抽象字符的一个串，而不隐含对它们进行编码的任何实现方式。

3.4.3 encoding algorithm 编码算法：一个精确的规范，规定了如何将一个使用规定字符的字符串高效地编码为八位组。

注 — 一个示例是将一个诸如"-32176"的串编码为一个包含两部分的二进制整数，并放在两个八位组中。这两个八位组的编码可以伴随一个标识了此编码算法的词汇表索引。

3.4.4 external vocabulary 外部词汇：由一个 URI 所引用的词汇表的一个集合（见 7.2.14 节）。

3.4.5 fast infoset document 快速信息集文档：按照本建议书 | 国际标准的规定来表示的一个 XML 信息集。

3.4.6 final vocabulary 最终词汇：在创建或处理一个快速信息集文档结束时，词汇表的内容。

3.4.7 information item 信息项：组成一个 XML 信息集的每种类型的项目。

3.4.8 initial vocabulary 初始词汇：根据一个快速信息集文档的头部信息而建立的词汇表的集合，可选的，它可以引用一个外部词汇并提供附加的表条目。

3.4.9 name surrogate 名字代理：用于表示一个限定名字（见 3.4.11）的三种词汇表索引（前两个是可选的）的集合。

3.4.10 namespace-well-formed XML document 命名空间良构的 XML 文档：或者是一个 W3C XML 1.0 文档，它根据 W3C XML 命名空间 1.0 被良好地构造；或者是一个 W3C XML 1.1 文档，它根据 W3C XML 命名空间 1.1 被良好地构造。

3.4.11 qualified name 限定名字：一个集合，由一个元素信息项或属性信息项的**[prefix]**，**[namespace name]** 和 **[local name]** 属性组成。

3.4.12 restricted alphabet 有限字母表：不同 ISO/IEC 10646 字符的一个有序集合，对于完全是由此集合中的字符构成的字符串可以进行一种压缩编码。

3.4.13 vocabulary table index 词汇表索引：一个正整数值，标识了某个词汇表中的一个条目。

3.4.14 vocabulary tables 词汇表：与某个快速信息集文档相关联的一系列概念表（典型地但不是必须地，这些表是动态构造的），这些表包含字符串或其他信息，并支持使用典型的小的正整数值（词汇表索引）来标识它们的条目。

注 — 词汇表的示例有：包含的字符串为属性或元素信息项的**[local name]**属性的那些表，或者包含的字符串是对应于系列**character**信息项的那些表，而这些**character**信息项是元素信息项的**[children]**属性的成员。

3.4.15 XML declaration XML 声明：一个规定字符串的 UTF-8 编码（也见 12.3 节），可能包含在一个快速信息集文档的起始处，用来标识此编码是一个快速信息集文档，并以此将其与一个 W3C XML 1.0 文档或 W3C XML 1.1 文档区别开来。

3.4.16 XML infoset XML 信息集：在一个命名空间良构的 XML 文档中，描述信息的一个抽象数据集，与 W3C XML 信息集中规定的相同。

3.4.17 XML whitespace XML 空白字符：Unicode 中的一个或多个下列字符：横向制表符（9），换行符（10），回车符（13），或空格（32）等。

注 — 这些字符是与W3C XML 1.0 和W3C XML 1.1（见W3C XML 1.0的2.3节和W3C XML 1.1的2.3节）中的产品"S"相匹配的那些字符。字符下一行（133）和行分隔符（8232），可能出现在一个命名空间良构的W3C XML 1.1文档中（见W3C XML 1.1的2.11节），通过对行结尾进行处理而被转换为行填充字符（见W3C XML 1.1的2.11节）。如果那些字符出现在一个由某个命名空间良构的W3C XML 1.1文档所产生的XML信息集中，则它们不是XML空白字符。

4 缩写

本建议书 | 国际标准采用下列缩写：

ASN.1	抽象句法符号1
BMP	基本多语言平台
ECN	编码控制符号
MIME	多用途因特网邮件扩展
UBL	统一商务语言
URI	通用资源标识符
UTF-8	通用转换功能8比特（见ISO/IEC 10646，附件D）
UTF-16BE	通用转换功能16比特大端（见Unicode，2.6）
UUID	通用唯一标识符
XML	可扩展标记语言

5 符号

5.1 本建议书 | 国际标准使用 ITU-T X.680 建议书 | ISO/IEC 8824-1 中定义的 ASN.1 符号来表示数据类型的正式定义，该数据类型的编码为快速信息集文档。

注 — 第12节规定了ITU-T X.692建议书 | ISO/IEC 8825-3到ASN.1类型定义的应用，提供了快速信息集文档的比特级编码。

5.2 在本建议书 | 国际标准中，使用**粗体 `Courier`** 来表示 ASN.1 符号，使用**粗体 `Arial`** 来表示 W3C XML 语法以及 XML 信息集中信息项的名字。

5.3 信息项的属性的名字使用**粗体 `Arial`**，并括在方括号之间（例如**`[children]`**）。

5.4 字符串类别的名字（见 8.4.2 节），以及限定名字类别的名字（见 8.5.4 节）使用大写字母。

5.5 在本建议书 | 国际标准中，一个八位组中的比特位置使用术语第一比特，第二比特，…，一直到第八比特来进行规范，其中第一比特是八位组中的最高有效位，而第八比特是八位组中的最低有效位。

6 词汇表构造和使用的原则

6.1 词汇表是一种概念上的表格，可以将词汇表索引映射为一个词汇表条目。

注 — 词汇表在计算机内存中的表示未定义，同样未定义的还有针对该表的词汇表索引到词汇表条目的映射实现方法。

6.2 由将 XML 信息集创建为一个快速信息集文档的创建者来决定词汇表的内容。

6.3 在更一般的情况下，一个快速信息集文档的头部可以引用一个词汇表的集合（一个外部词汇），后面跟一个规范，该规范规定了如何通过附加到这些词汇表上来为此快速信息集文档构建初始词汇。在创建和处理一个快速信息集文档的过程中，还会更进一步地附加到词汇表上，因此词汇表会不断地增长，形成该文档的最终词汇表。

6.4 在创建和处理一个快速信息集文档的过程中，某些词汇表从初始词汇到最终词汇会有很大的增长，因此在词汇表的名称中有“动态”一词。没有为从任意表中删除条目而定义机制。

6.5 词汇表索引是被隐含分配的。任何词汇表的第一个条目，其词汇表索引都为 1，且每个后续的表条目都具有下一个顺序的词汇表索引。由于本建议书 | 国际标准中规定了会有某些内容加入到词汇表中，这就意味着必须分配下一个可用的词汇表索引。

注 — 词汇表索引起始于1，且不为0，因为0值（如果允许的话）在一个字段中具有特殊的含义，表示“空字符串”，而此字段另外可能还拥有一个词汇表索引。

6.6 为了支持词汇表索引的这种隐含分配，处理一个快速信息集文档的组件（在任意深度）的概念顺序应被完整地定义（见 8.1 节）。

注 — 此顺序与对快速信息集文档中的组件进行编码的顺序是相同的。但它并没有隐含说明此文档所携带的语义也要按照此顺序来进行处理。定义此顺序的目的仅仅是要确保一个快速信息集文档的创建者和处理者对任意给定的词汇表条目所分配的词汇表索引是一致的。

6.7 词汇表可被用于很多目的（见第 8 节），但是它们的主要功能是要使用一个词汇表索引来替代一个词汇表条目，这些索引比表条目要更小（而且可能处理上更快）。为某些词汇表定义的一定数量的内置条目在第 9 节规定。这些条目总是隐含地出现在这些词汇表中，并具有第 9 节所规定的词汇表索引。

6.8 对于某些类型的字符串，快速信息集文档的创建者可以选择向一个词汇表中增加或不增加该字符串，这是依赖于该字符串在 XML 信息集中期望的（或已知的）出现次数。

6.9 词汇表条目的确切构造和含义在第 8 节中规定，但在大多数情况下，它们都是可变长度的字符串，一般短于，但潜在的最大长度为 2^{32} 个八位组。

6.10 向 7.13.7 节，7.14.6 节，7.14.7 节和 7.16.7 节中规定的词汇表进行附加的所有工作，都要求由一个快速信息集文档的相容创建者来执行。这就确保了每个词汇表中的词汇表条目的数量永远不会超出 2^{20}。

注 — 一个词汇表条目可能等于一个或多个其他词汇表条目。这是为了允许高效地创建快速信息集文档。然而，重复的条目将降低传送效率。处理者不受重复条目的影响。

6.11 向 7.13.8 节，7.14.11 节和 7.16.8 节中规定的词汇表进行附加的所有工作，要求由一个快速信息集文档的相容处理者来执行。这就确保了不会违反 6.10 的限制。

7 ASN.1类型定义

7.1 概要

7.1.1 本建议书 | 国际标准规定了用以支持 XML 信息集表示的一系列 ASN.1 类型。这一系列类型的根类型为 **Document** 类型。

7.1.2 为了改善规范的可用性以及由此产生的编码的有效性，对 XML 信息集的内容施加了一些限制，并且在表示中作了一些简化（见第 11 节）。

注 — 一个不符合上述限制的XML信息集不能被表示为一个快速信息集文档，也不能被表示为一个命名空间良构的XML文档。

7.1.3 针对 W3C XML 信息集中规定的每种信息项类型，在本建议书 | 国际标准中都提供了一种相应的 ASN.1 类型定义。这种类型定义总是一个序列（sequence）类型，其中的组件对应于信息项中的属性。

7.1.4 信息项中的某些属性没有包含在 ASN.1 类型定义中（见 11.4 节）。

7.1.5 在某些情况下，没有包含在 ASN.1 类型定义中的某个属性的值，可以通过包含在 ASN.1 类型定义内的同一个信息项或不同信息项中的其他属性的值推导出来。在这些情况下，忽略此属性可以简化表示，同时也不丢失信息。然而，还是有一些比较少的情况，没有包含在内的某个属性的值并不能够通过其他属性推导出来。在所有上述情况下，忽略此属性是一种简化方法，而且在绝大多数实际用例中并没有对规范的有效性有所限制。

7.1.6 第 12 节规定了 **Document** 类型的编码。

7.2 Document类型

7.2.1 **Document** 类型的定义为：

```
Document ::= SEQUENCE {
      additional-data          SEQUENCE (SIZE(1..one-meg)) OF
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } OPTIONAL,
      initial-vocabulary       SEQUENCE {
            external-vocabulary         URI OPTIONAL,
            restricted-alphabets        SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            encoding-algorithms         SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            prefixes                    SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            namespace-names             SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            local-names                 SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-ncnames               SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-uris                  SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            attribute-values            SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            content-character-chunks    SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            other-strings               SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            element-name-surrogates     SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL,
            attribute-name-surrogates   SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL }
                  (CONSTRAINED BY {
                        -- 如果初始词汇组件存在，
                        -- 则至少应当有它的一个组件存在 -- }) OPTIONAL,
      notations                     SEQUENCE (SIZE(1..MAX)) OF
                                          Notation OPTIONAL,
      unparsed-entities             SEQUENCE (SIZE(1..MAX)) OF
                                          UnparsedEntity OPTIONAL,
```

```
character-encoding-scheme  NonEmptyOctetString OPTIONAL,
standalone                 BOOLEAN OPTIONAL,
version                    NonIdentifyingStringOrIndex OPTIONAL
                             -- OTHER STRING 类别 --,
children                   SEQUENCE (SIZE(0..MAX)) OF
      CHOICE {
            element                    Element,
            processing-instruction     ProcessingInstruction,
            comment                    Comment,
            document-type-declaration DocumentTypeDeclaration }}
```

其中 **one-meg** 的值为：

```
one-meg INTEGER ::= 1048576 - 2 的 20 次方
```

NonEmptyOctetString 类型定义为：

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

four-gig 的值为：

```
four-gig INTEGER ::= 4294967296 - 2 的 32 次方
```

URI 类型定义为：

```
URI ::= NonEmptyOctetString
```

7.2.2 **EncodedCharacterString**，**NameSurrogate**，**Notation**，**UnparsedEntity**，**NonIdentifyingStringOrIndex**，**Element**，**ProcessingInstruction**，**Comment** 和 **DocumentTypeDeclaration** 类型分别在 7.17 节，7.15 节，7.11 节，7.10 节，7.14 节，7.3 节，7.5 节，7.8 节和 7.9 节定义。

7.2.3 **URI** 类型应当是 IETF RFC 2396 中规定的一个 URI。

7.2.4 **initial-vocabulary** 中的 **restricted-alphabets** 组件（如果存在的话）应当携带一个或多个字符串，其中每个字符串都带有某个有限字母表中的字符。每个字符串应当至少包含两个字符，且字符串中的所有字符都应当是不同的。

注 — 使用有限字母表来优化对字符串的编码在7.17.6节中规定。

7.2.5 **initial-vocabulary** 中的 **encoding-algorithms** 组件(如果存在的话)应当携带一个或多个 URI，其中每个 URI 都标识一种编码算法。

注 — 在本建议书 | 国际标准中定义了内置的编码算法（见第10节），以及为其规定的词汇表索引，但是定义更进一步的编码算法及其相关的URI不在本建议书 | 国际标准的定义范围之内，同时定义这里所确定的这些算法的方式也不在本建议书 | 国际标准的定义范围之内。定义一个编码算法所需的信息在第8.3.3节中规定。

7.2.6 **Document** 类型表示一个 XML 信息集的 **document** 信息项。由于一个 XML 信息集中的所有其他信息项都或者是此信息项的属性，或者是此信息项的子类或后代的属性（在任意深度），因此每个 **Document** 都表示一个完整的 XML 信息集。

注 — 每个**Document**，如果没有引用一个外部词汇（见7.2.13节），则它也定义了一个最终词汇，可被用作某些其他快速信息集文档的外部词汇。

7.2.7 组件 **additional-data**（如果存在的话）应当携带一个或多个 **additional-datum** 组件，来允许使用附加机制处理快速信息集文档。

注 1 — 一个示例是这样一种数据，即可以使一个处理者能够访问某个快速信息集文档的一部分，而不需要处理整个文档。这种数据的格式未定义。

注 2 — 组件**additional-datum**的数量限制在2^{20} 个组件之内（见7.2.1节）。

7.2.8 每个 **additional-datum** 应当包括：

a） **id**组件（URI类型的一个值）：此URI应当引用一个规范，该规范定义了**data**组件的格式和语义；以及

注 — **additional-datum**的格式可能会被规定为一个抽象类型再加上一个编码规则，或者采用其他任何适当的方式。

b） **data**组件，这是一个八位组串，携带附加的处理数据。

7.2.9 一个 **additional-data** 组件的用法受下列情形的影响：

a） 一个**additional-datum**组件能够被某个处理者忽略，除非该URI是被公认的，且附加的处理被认为是与该处理者的活动相关的；

b） 一个忽略所有**additional-datum** 组件的处理者仍有能力产生一个XML信息集，此信息集与用来产生快速信息集文档的XML信息集是等同的。

7.2.10 可能会存在具有相同 URI 的多个 **additional-datum** 组件，并且将根据与此 URI 相关的规范来处理这些组件。

7.2.11 组件 **initial-vocabulary** 可以提供数据（与某些内置的表项一起）来完全决定此快速信息集文档的有限字母表（见 8.2 节），编码算法表（见 8.3 节），动态串表（见 8.4 节），以及动态名字表（见 8.5 节）等的初始内容（快速信息集文档的初始词汇）。一个初始词汇包含如下数据：

a） 有限字母的一个有序集合（见8.2.2），至少包含内置的有限字母（见第9节）；

b） 编码算法的一个有序集合（见8.3.2），至少包含内置的编码算法（见第10节）；

c） 字符串的八个独立的有序集合，对应于本建议书 | 国际标准中规定的字符串的八种类别（见8.4.2），其中每个集合都包含零个或多个单一类别的字符串；以及

d） 名字代理的两个独立的有序集合（见8.5.2），对应于本建议书 | 国际标准中规定的限定名字的两种类别（见8.5.4），其中每个集合都包含零个或多个单一类别的名字代理。

注 — 一个初始词汇不能完全是空的，因为它总是（至少）包含内置的有限字母表和内置的编码算法。然而，对于一个快速信息集文档来说，它所具有的初始词汇如果仅包含那些数据也并不少见，因为（由一个快速信息集文档的创建者作出的）关于如何使用此**initial-vocabulary**组件的决定是依赖于实现的，而且某些实现可能会选择动态地增加所有的词汇表条目（在快速信息集文档内）。

7.2.12 快速信息集文档的初始词汇应当根据如下情况来判定：

a） 如果**initial-vocabulary**组件缺失，则初始词汇应当仅包含7.2.21节、7.2.22节、第9节和第10节中规定的内置表条目。

b） 如果**initial-vocabulary**组件存在，但**external-vocabulary**组件缺失，则初始词汇应当包含7.2.21节、7.2.22节、第9节和第10节中规定的内置表条目，以及由7.2.16节决定的附加表条目（如果有的话）。

c） 如果**initial-vocabulary**组件存在，且**external-vocabulary**组件也存在，则初始词汇应当包含由7.2.13节和7.2.14节所规定的**external-vocabulary**组件所标识的最终词汇，以及由7.2.16节决定的附加表条目（如果有的话）。

7.2.13 **external-vocabulary** 组件使用 7.2.14 节所规定的机制之一标识了一个最终词汇。URI 类型（见 7.2.1 节）通过三种方法中的其中一种决定了要用作外部词汇的最终词汇（见 7.2.14 节）。

注 — 本建议书 | 国际标准不规定任何外部词汇和任何引用外部词汇的URI。这些外部词汇和URI可以由任何能够分配URI的当局来定义，并且可能是私有协商的，也可能是需要标准化的。

7.2.14 一个外部词汇可以根据下列三种方法之一来指定：

a） 是一个快速信息集文档的最终词汇，其本身不能再引用一个外部词汇，或者：

注 1 — 最终词汇是否在本地存储，或者是否仅有快速信息集文档被存储，而最终词汇是通过对其进行处理而产生的等等，这些都是在实现时考虑的内容。

注 2 — 一个快速信息集文档的最终词汇，如果引用了一个外部词汇，则它本身不能再被用作一个外部词汇，施加这种限制是为了简化实现，并避免引用的循环。

b） 是一个命名空间良构的XML文档，它做了如下概念上的处理：

1） 命名空间良构的XML文档的XML信息集应当是已决定的；且

2） 该XML信息集的快速信息集文档应当根据本建议书 | 国际标准中规定的那样创建，但它不应当具有 **initial-vocabulary** 组件，且 **NonIdentifyingStringOrIndex** 的 **add-to-table**组件（见7.14）应当总是被赋值为**TRUE**，并且在任何一个串表中都不应当出现多个相同的字符串；且

3） 此快速信息集文档的最终词汇变为外部词汇；或者

注 3 — 最终词汇是否在本地存储，或者是否仅有 XML 文档被存储，而最终词汇是通过对其进行处理而产生的等等，这些都是在实现时考虑的内容。

c） 是使用任何其他足够精确的机制或文本所规定的词汇表的集合，其中应当包含第9节和第10节的内置表条目（以及这些章节中规定的词汇表索引）。

注 4 — 规定定义词汇表时使用的符号不在本建议书 | 国际标准的定义范围之内。

注 5 — 在使用这种机制时要求包含内置表条目，就确保了所有的词汇表都包含了内置表条目。

7.2.15 对于一个根据7.2.14 c)规定的外部词汇，其所有的串和名字表条目，除了那些在PREFIX表和NAMESPACE NAME表中的条目以外，都应当被分配一个起始为 1 的连续索引。PREFIX 表和 NAMESPACE NAME 表中的条目应当被分配一个起始为 2 的连续索引。所有的有限字母，除了内置的以外，都应当被分配一个起始为 16 的连续索引。所有的编码算法，除了内置的以外，都应当被分配一个起始为 32 的连续索引。

7.2.16 每一个出现在 **initial-vocabulary** 的其余组件中的 **NonEmptyOctetString**，**EncodedCharacterString** 和 **NameSurrogate**（如果有的话）都应当被按顺序（见 8.1 节）加入到一个词汇表中，如表 1 的规定。

表 1－组件标识符到词汇表的映射

组件标识符	条目的ASN.1类型	词汇表（见第8节）
restricted-alphabets	**NonEmptyOctetString**	有限字母表（见 8.2 节）
encoding-algorithms	**NonEmptyOctetString**	编码算法表（见 8.3 节）
prefixes	**NonEmptyOctetString**	PREFIX 表（见 8.4 节）
namespace-names	**NonEmptyOctetString**	NAMESPACE NAME 表（见 8.4 节）
local-names	**NonEmptyOctetString**	LOCAL NAME 表（见 8.4 节）
other-ncnames	**NonEmptyOctetString**	OTHER NCNAME 表（见 8.4 节）
other-uris	**NonEmptyOctetString**	OTHER URI 表（见 8.4 节）
attribute-values	**EncodedCharacterString**	ATTRIBUTE VALUE 表（见 8.4 节）
content-character-chunks	**EncodedCharacterString**	CONTENT CHARACTER CHUNK 表（见 8.4 节）
other-strings	**EncodedCharacterString**	OTHER STRING 表（见 8.4 节）
element-name-surrogates	**NameSurrogate**	ELEMENT NAME 表（见 8.5 节）
Attribute-name-surrogates	**NameSurrogate**	ATTRIBUTE NAME 表（见 8.5 节）

7.2.17 **NonEmptyOctetString** 类型的值应当携带一个字符串的 UTF-8 编码（见 ISO/IEC 10646，附件 D）。

7.2.18 一个初始词汇中的有限字母表和编码算法表应当最多有 256 个条目。所有其他表应当最多有 2^{20} 个条目。

注 — 对条目数量的限制是为了确保表索引的公共上界。如果表条目是被动态加入的也应当应用此限制（见7.13.7节，7.14.6节，7.14.7节和7.16.7节）。这些限制不会阻止将任何XML信息集编码为一个快速信息集文档。

7.2.19 内置的有限字母具有 1 和 2 之间的词汇表索引（见第 9 节）。在 **initial-vocabulary**（如果存在的话）的 **restricted-alphabets** 组件内的有限字母表的词汇表索引应当按照如下规定分配：

a） 如果没有外部词汇，或者外部词汇仅包含内置的有限字母表，则索引应当从16开始分配；

b） 否则，索引应当从外部词汇中最高的有限字母表索引再加上1开始分配。

注 — 这就意味着词汇表索引3到15没有被使用。这些值为本建议书 | 国际标准的后续版本所预留。

7.2.20 内置的编码算法具有 1 到 10 之间的词汇表索引（见第 10 节）。在 **initial-vocabulary**（如果存在的话）的 **encoding-algorithms** 组件内的编码算法的词汇表索引应当按照如下规定分配：

a） 如果没有外部词汇，或者外部词汇仅包含内置的编码算法，则索引应当从32开始分配；

b） 否则，索引应当从外部词汇中最高的编码算法索引再加上1开始分配。

注 — 这就意味着词汇表索引11到31没有被使用。这些值为本建议书 | 国际标准的后续版本所预留。

7.2.21 PREFIX 表应当具有一个内置的前缀条目"xml"，为其分配索引为 1。在 **initial-vocabulary**（如果存在的话）的 **prefixes** 组件内的前缀的词汇表索引应当按照如下规定分配：

a） 如果没有外部词汇，或者外部词汇仅包含内置的前缀条目，则索引应当从2开始分配；

b） 否则，索引应当从外部词汇中最高的前缀索引再加上1开始分配。

7.2.22 NAMESPACE NAME 表应当具有一个内置的命名空间名字条目：

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

应当为其分配索引 1。

7.2.23 在 **initial-vocabulary**（如果存在的话）的 **namespace-names** 组件内的命名空间名字的词汇表索引应当按照如下规定分配：

a） 如果没有外部词汇，或者外部词汇仅包含内置的命名空间名字条目，则索引应当从2开始分配；

b） 否则，索引应当从外部词汇中最高的命名空间名字索引再加上1开始分配。

7.2.24 组件 **notations** 表示 **document** 信息项的**[notations]** 属性。该组件的类型是一个 sequence-of 类型，尽管在 W3C XML 信息集中，**[notations]** 属性是被规定为一个（由 **notation** 信息项组成的）无序数组。

注 — 在这里以及其他地方，使用sequence-of类型比set-of类型更合适一些，因为后一种类型不能满足对一个快速信息集文档中所有组件进行严格排序的需求（见8.1节）。

7.2.25 组件 **unparsed-entities** 表示 **document** 信息项的**[unparsed entities]** 属性。该组件的类型是一个 sequence-of 类型，尽管在 W3C XML 信息集中，**[unparsed entities]** 属性是被规定为一个（由 **unparsed entity** 信息项组成的）无序数组。

7.2.26 组件 **character-encoding-scheme** 表示 **document** 信息项的**[character encoding scheme]** 属性。该组件的类型是 **NonEmptyOctetString**，且该组件的值应当携带**[character encoding scheme]**属性的 UTF-8 编码（见 ISO/IEC 10646，附件 D）。在 **Document** 类型的某个抽象值中如果缺少该组件，则表示**[character encoding scheme]** 属性取值为"UTF-8"。

注 — 支持**[character encoding scheme]** 属性可以使得XML文档与快速信息集文档之间可以进行相互转换，而不会改变字符的编码模式。从一个XML文档创建一个快速信息集文档的创建者可能会对**[character encoding scheme]**属性进行编码，该属性是从XML文档的编码声明中获得的（见W3C XML 1.0的4.3.1节和W3C XML 1.1的4.3.1节）。一个快速信息集文档的处理者可以使用**character-encoding-scheme**组件（如果存在的话），如果它希望产生初始编码的话。

7.2.27 组件 **standalone** 表示 **document** 信息项的**[standalone]** 属性。抽象值 **TRUE** 表示该属性的值为 **yes**，抽象值 **FALSE** 表示其值为 **no**。在 **Document** 类型的某个抽象值中如果缺少该组件，则表示**[standalone]** 属性无值。

7.2.28 组件 **version** 表示 **document** 信息项的 **[version]** 属性。该组件的类型是 **NonIdentifyingStringOrIndex**（见 7.14 节），在这里表示为一个 OTHER STRING 类别的字符串。在 **Document** 类型的某个抽象值中如果缺少该组件，则表示 **[version]** 属性无值。

7.2.29 组件 **children** 表示 **document** 信息项的**[children]** 属性。在 sequence-of 中，有且仅有一个项（在任意位置）应当使用 choice 类型中的选择项 **element**，且最多有一个项（在任意位置）应当使用选择项 **document-type-declaration**。其他每个项（如果有的话）应当或者使用选择项 **processing-instruction**，或者使用选择项 **comment**。

7.2.30 **document** 信息项中的**[document element]** 属性没有包含在 **Document** 类型中。该属性的值总是一个作为 **document** 信息项中**[children]**属性的一个成员的唯一 **element** 信息项。

7.2.31 **document** 信息项中的**[base URI]** 属性没有包含在 **Document** 类型中，且不被本建议书 | 国际标准所支持。

7.2.32 **document** 信息项中的**[all declarations processed]**属性没有包含在 **Document** 类型中，且假设取值为 **true** （见 11.3 节）。

7.3 Element类型

7.3.1 **Element** 类型的定义为：

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

7.3.2 **NameSpaceAttribute**，**QualifiedNameOrIndex**，**Attribute**，**ProcessingInstruction**，**UnexpandedEntityReference**，**CharacterChunk** 和 **Comment** 类型分别在 7.12 节，7.16 节，7.4 节，7.5 节，7.6 节，7.7 节和 7.8 节定义。

7.3.3 **Element** 类型表示 XML 信息集的元素（**element** ）信息项。

7.3.4 组件 **namespace-attributes** 表示 **element** 信息项的**[namespace attributes]** 属性。该组件的类型是一个 sequence-of 类型，尽管在 W3C XML 信息集中，**[namespace attributes]** 属性被规定为是一个（由 **attribute** 信息项组成的）无序数组。

注 — sequence-of的组件的类型为**NamespaceAttribute**（而不是**Attribute**），尽管在W3C XML信息集中，元素信息项中的**[namespace attributes]** 属性被规定为是一个由**attribute**信息项组成的集合。在一个有限的XML信息集中（见11.3节），一个**namespace** 信息项的属性可以通过一个表示命名空间属性的**attribute**信息项的属性来决定。相反的情况只有部分是正确的，但是这种限制在本建议书 | 国际标准的预期使用中被认为是可接受的。（也见7.2.24节的注）。

7.3.5 组件 **qualified-name** 表示 **element** 信息项的限定名字（见 3.4.11）（即包含该信息项的 **[prefix]**，**[namespace name]**和**[local name]** 属性的集合）。该组件的类型为 **QualifiedNameOrIndex**（见 7.16 节），在这里表示 ELEMENT NAME 类别的一个限定名字。

7.3.6 组件 **attributes** 表示 **element** 信息项的**[attributes]** 属性。该组件的类型是一个 sequence-of 类型，尽管在 W3C XML 信息集中，**[attributes]** 属性被规定为是一个（由 **attribute** 信息项组成的）无序数组。

7.3.7 组件 **children** 表示 **element** 信息项的**[children]** 属性。当有两个或多个相邻的 children 是 **character** 信息项，则可能使用一个单独的 **CharacterChunk** 项来表示那些相邻的 **character** 信息项。

注 — 如果在一个**element**信息项的children内，有N个相邻的字符组成了一个序列，则允许这N个字符任意分组为一系列连续的字符块。然而，为了进行有效的编码，希望快速信息集文档的创建者能够让每个字符块尽可能大。

7.3.8 **element** 信息项的**[in-scope namespaces]** 属性没有包含在 **Element** 类型中。

注 — 在一个有限的XML信息集中（见11.3节），一个**element**信息项的**[in-scope namespaces]** 属性能够通过**element**信息项的**[namespace attributes]**属性，以及（直接或间接）包含该**element**信息项的所有其他**element**信息项（如果有的话）的**[namespace attributes]**属性来决定。

7.3.9 **element** 信息项的**[base URI]** 属性没有包含在 **Element** 类型中，并且不被本建议书 | 国际标准所支持。

7.3.10 **element** 信息项的**[parent]** 属性没有包含在 **Element** 类型中。对于一个任意给定的 **element** 信息项，该属性的值是包含该信息项，并将该信息项作为其**[children]**属性的一个成员的**document**信息项或**element** 信息项。

7.4 Attribute类型

7.4.1 **Attribute** 类型的定义为：

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

7.4.2 **QualifiedNameOrIndex** 和 **NonIdentifyingStringOrIndex** 类型分别在 7.16 节和 7.14 节定义。

7.4.3 **Attribute** 类型表示 XML 信息集的属性（**attribute**）信息项。

7.4.4 组件 **qualified-name** 表示 **attribute** 信息项的限定名字（见 3.4.11 节）（即包含该信息项的 **[prefix]**，**[namespace name]**和**[local name]** 属性的集合）。该组件的类型为 **QualifiedNameOrIndex**（见 7.16 节），在这里表示 ATTRIBUTE NAME 类别的一个限定名字。

7.4.5 组件 **normalized-value** 表示 **attribute** 信息项的**[normalized value]** 属性。该组件的类型为 **NonIdentifyingStringOrIndex**（见 7.14 节），在这里表示 ATTRIBUTE VALUE 类别的一个字符串。

7.4.6 分配给 **normalized-value** 的字符串长度不能大于 2^{32}。

注 — 这个限制是由ASN.1的定义所隐含的，设计该限制是为了优化编码并简化实现（也见11.3 j）。

7.4.7 **attribute** 信息项的**[specified]** 属性没有包含在 **Attribute** 类型中。

7.4.8 **attribute** 信息项的**[attribute type]** 属性没有包含在 **Attribute** 类型中。

7.4.9 **attribute** 信息项的**[references]** 属性没有包含在 **Attribute** 类型中。

注 — 在一个有限的XML信息集中（见11.3节），一个**attribute**信息项的**[references]** 属性能够通过**attribute**信息项的**[normalized value]**属性，以及XML信息集中的其他信息项的属性来决定。

7.4.10 **attribute** 信息项的**[owner element]** 属性没有包含在 **Attribute** 类型中。对于一个任意给定的 **attribute** 信息项，该属性的值是包含此信息项，并将此信息项作为其**[attributes]**属性的一个成员的 **element** 信息项。

7.5 ProcessingInstruction类型

7.5.1 **ProcessingInstruction** 类型的定义为：

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

7.5.2 **IdentifyingStringOrIndex** 和 **NonIdentifyingStringOrIndex** 类型分别在 7.13 节和 7.14 节定义。

7.5.3 **ProcessingInstruction** 类型表示 XML 信息集的处理指示（**processing instruction**）信息项。

7.5.4 组件 **target** 表示 **processing instruction** 信息项的 **[target]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER NCNAME 类别的一个字符串。

7.5.5 组件 **content** 表示 **processing instruction** 信息项的 **[content]** 属性。该组件的类型为 **NonIdentifyingStringOrIndex**（见 7.14 节），在这里表示 OTHER STRING 类别的一个字符串。

7.5.6 分配给 **content** 的字符串长度不能大于 2^{32}。

注 — 这个限制是由ASN.1的定义所隐含的，设计该限制是为了优化编码并简化实现（也见11.3 j）。

7.5.7 **processing instruction** 信息项的**[notation]** 属性没有包含在 **ProcessingInstruction** 类型中。

注 — 在一个有限的XML信息集中（见11.3节），一个**processing instruction**信息项的[**notation**]属性能够通过**processing instruction**信息项的[**target**]属性，以及**document**信息项中的[**notations**]属性来决定。

7.5.8 **processing instruction** 信息项的**[parent]** 属性没有包含在 **ProcessingInstruction** 类型中。对于一个任意给定的 **processing instruction** 信息项，该属性的值是包含此信息项，并将此信息项作为其**[children]** 属性的一个成员的 **document** 信息项，**element** 信息项或 **document type definition** 信息项。

7.6 UnexpandedEntityReference 类型

7.6.1 **UnexpandedEntityReference** 类型的定义为：

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

7.6.2 **IdentifyingStringOrIndex** 类型在 7.13 节定义。

7.6.3 **UnexpandedEntityReference** 类型表示 XML 信息集的未展开的实体引用（**unexpanded entity reference**）信息项。

7.6.4 组件 **name** 表示 **unexpanded entity reference** 信息项的**[name]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER NCNAME 类别的一个字符串。

7.6.5 组件 **system-identifier** 表示 **unexpanded entity reference** 信息项的**[system identifier]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER URI 类别的一个字符串。在 **UnexpandedEntityReference** 类型的一个抽象值中；如果该组件缺失则表示**[system identifier]**属性无值。

7.6.6 组件 **public-identifier** 表示 **unexpanded entity reference** 信息项的**[public identifier]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER URI 类别的一个字符串。在 **UnexpandedEntityReference** 类型的一个抽象值中，如果该组件缺失则表示**[public identifier]**属性无值。

7.6.7 **unexpanded entity reference** 信息项的**[declaration base URI]** 属性没有包含在 **UnexpandedEntityReference** 类型中，且不被本建议书 | 国际标准所支持。

7.6.8 **unexpanded entity reference** 信息项的**[parent]** 属性没有包含在 **UnexpandedEntityReference** 类型中。对于一个任意给定的 **unexpanded entity reference** 信息项，该属性的值是包含此信息项，并将此信息项作为其**[children]**属性的一个成员的 **element** 信息项。

7.7 CharacterChunk 类型

7.7.1 **CharacterChunk** 类型的定义为：

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

7.7.2 **NonIdentifyingStringOrIndex** 类别在 7.14 节中定义。

7.7.3 **CharacterChunk** 类型对应于 **character** 信息项，但表示一系列相邻的 **character** 信息项（父亲 **element** 信息项中 **[children]** 的成员），而不是一个单独的 **character** 信息项。

7.7.4 由 **CharacterChunk** 类型的一个值来表示的 **character** 信息项的数量不能为 0。

7.7.5 组件 **character-codes** 表示块中（多个）**character** 信息项的**[character code]** 属性。该组件的类型为 **NonIdentifyingStringOrIndex**（见 7.14 节），在这里表示 CONTENT CHARACTER CHUNK 类别的一个字符串。

7.7.6 分配给 **character-codes** 的字符串的长度不能大于 2^{32}。

注 — 这个限制是由ASN.1的定义所隐含的，设计该限制是为了优化编码并简化实现。这种限制不会阻止一个包含多于2^{32} 个**character**信息项的**element**信息项的编码，因为可以使用多个块。

7.7.7 **character** 信息项的**[element content whitespace]** 属性没有包含在 **CharacterChunk** 类型中。

7.7.8 **character**信息项的**[parent]** 属性没有包含在**CharacterChunk** 类型中。对于一个任意给定的**character**信息项，该属性的值是包含此信息项，并将此信息项作为其**[children]**属性的一个成员的 **element** 信息项。

7.8 Comment 类型

7.8.1 **Comment** 类型的定义为：

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

7.8.2 **NonIdentifyingStringOrIndex** 类型在 7.14 节中定义。

7.8.3 **Comment** 类型表示 XML 信息集的注释（**comment**）信息项。

7.8.4 组件 **content** 表示 **comment** 信息项的 **[content]** 属性。该组件的类型为 **NonIdentifyingStringOrIndex** 类型（见 7.14 节），在这里表示 OTHER STRING 类别的一个字符串。

7.8.5 分配给 **content** 的字符串的长度不能大于 2^{32}。

注 — 这个限制是由ASN.1的定义所隐含的，设计该限制是为了优化编码并简化实现（也见11.3 j）。

7.8.6 **comment** 信息项的**[parent]** 属性没有包含在 **Comment** 类型中。对于一个任意给定的 **comment** 信息项，该属性的值是包含此信息项，并将此信息项作为其**[children]**属性的一个成员的 **document** 信息项或 **element** 信息项。

7.9 DocumentTypeDeclaration 类型

7.9.1 **DocumentTypeDeclaration** 类型的定义为：

```
DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier    IdentifyingStringOrIndex OPTIONAL
                               -- OTHER URI 类别 --,
      public-identifier    IdentifyingStringOrIndex OPTIONAL
                               -- OTHER URI 类别 --,
      children             SEQUENCE (SIZE(0..MAX)) OF
            ProcessingInstruction }
```

7.9.2 **IdentifyingStringOrIndex** 类型在 7.13 节中定义。

7.9.3 **DocumentTypeDeclaration** 类型表示 XML 信息集的文档类型声明（**document type declaration**）信息项。

7.9.4 组件 **system-identifier** 表示 **document type declaration** 信息项的**[system identifier]** 属性。该组件的类型为 **IdentifyingStringOrIndex** （见 7.13 节），在这里表示 OTHER URI 类别的一个字符串。在 **DocumentTypeDeclaration** 类型的某个抽象值中，如果该组件缺失则表示**[system identifier]**属性无值。

7.9.5 组件 **public-identifier** 表示 **document type declaration** 信息项的**[public identifier]**属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER URI 类别的一个字符串。在 **DocumentTypeDeclaration** 类型的某个抽象值中，如果该组件缺失则表示**[public identifier]**属性无值。

7.9.6 组件 **children** 表示 **document type declaration** 信息项的**[children]**属性。

7.9.7 **document type declaration** 信息项的**[parent]**属性没有包含在 **DocumentTypeDeclaration** 类型中。对于一个任意给定的 **document type declaration** 信息项，该属性的值是包含此信息项，并将此信息项作为其**[children]**属性的一个成员的 **document** 信息项。

7.10 UnparsedEntity 类型

7.10.1 **UnparsedEntity** 类型的定义为：

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

7.10.2 **IdentifyingStringOrIndex** 类型在 7.13 节中定义。

7.10.3 **UnparsedEntity** 类型表示 XML 信息集的未解析实体（**unparsed entity**）信息项。

7.10.4 组件 **name** 表示 **unparsed entity** 信息项的 **[name]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER NCNAME 类别的一个字符串。

7.10.5 组件 **system-identifier** 表示 **unparsed entity** 信息项的**[system identifier]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER URI 类别的一个字符串。

7.10.6 组件 **public-identifier** 表示 **unparsed entity** 信息项的**[public identifier]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER URI 类别的一个字符串。在 **UnparsedEntity** 类型的某个抽象值中，如果该组件缺失，则表示**[public identifier]** 属性无值。

7.10.7 组件 **notation-name** 表示 **unparsed entity** 信息项的**[notation name]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER NCNAME 类别的一个字符串。

7.10.8 **unparsed entity** 信息项的**[declaration base URI]** 属性没有包含在 **UnparsedEntity** 类型中，且不被本建议书 | 国际标准所支持。

7.10.9 **unparsed entity** 信息项的**[notation]** 属性没有包含在 **UnparsedEntity** 类型中。

注 — 在一个有限的XML信息集中（见11.3节），一个**unparsed entity**信息项的**[notation]**属性能够通过该**unparsed entity**信息项的**[notation name]**属性，以及**document**信息项的**[notations]**属性来决定。

7.11 Notation 类型

7.11.1 **Notation** 类型的定义为：

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

7.11.2 **IdentifyingStringOrIndex** 类型在 7.13 节定义。

7.11.3 **Notation** 类型表示 XML 信息集的符号（**notation**）信息项。

7.11.4 组件 **name** 表示 **notation** 信息项的**[name]**属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER NCNAME 类别的一个字符串。

7.11.5 组件 **system-identifier** 表示 **notation** 信息项的**[system identifier]**属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER URI 类别的一个字符串。在 **Notation** 类型的某个抽象值中，如果该组件缺失，则表示**[system identifier]** 属性无值。

7.11.6 组件 **public-identifier** 表示 **notation** 信息项的**[public identifier]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 OTHER URI 类别的一个字符串。在 **Notation** 类型的某个抽象值中，如果该组件缺失，则表示**[public identifier]** 属性无值。

7.11.7 **notation** 信息项的**[declaration base URI]** 属性没有包含在 **Notation** 类型中，且不被本建议书 | 国际标准所支持。

7.12 NamespaceAttribute 类型

7.12.1 **NamespaceAttribute** 类型的定义为：

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

7.12.2 **IdentifyingStringOrIndex** 类型在 7.13 节定义。

7.12.3 **NamespaceAttribute** 类型表示 XML 信息集中，作为某个 **element** 信息项中**[namespace attributes]** 属性的一个成员的属性（**attribute**）信息项。

注 — 在XML信息集中，属性和命名空间属性都是**attribute**信息项。在本建议书 | 国际标准中，使用不同的类型是为了进行优化。

7.12.4 在 XML 信息集中，有两种类型的命名空间属性：

a） 缺省的命名空间声明：**attribute**信息项的**[prefix]** 属性无值，且**[local name]**属性取值为**"xmlns"**；

b） 非缺省的命名空间声明：**attribute**信息项的**[prefix]** 属性取值为**"xmlns"**，且**[local name]**属性提供了命名空间声明的前缀。

在这两种情况下，**attribute** 信息项的**[normalized value]**属性都提供了命名空间声明的命名空间名字。

7.12.5 如果命名空间属性是一个缺省的命名空间声明（7.12.4 中的 a 情况），则组件 **prefix** 应当缺失，否则（7.12.4 中的 b 情况）该组件应当存在，表示 **attribute** 信息项的**[local name]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 PREFIX 类别的一个字符串。

7.12.6 如果 **attribute** 信息项的**[normalized value]** 属性取值为一个空串，则组件 **namespace-name** 应当缺失；否则该组件应当存在，表示 **attribute** 信息项的**[normalized value]** 属性。该组件的类型为 **IdentifyingStringOrIndex**（见 7.13 节），在这里表示 NAMESPACE NAME 类别的一个字符串。

7.12.7 **attribute** 信息项的**[namespace name]** 属性的取值总是**"http://www.w3.org/2000/xmlns/"**（见 W3C XML 信息集），且没有包含在 **NamespaceAttribute** 类型中。

7.13 IdentifyingStringOrIndex 类型

7.13.1 **IdentifyingStringOrIndex** 类型定义为：

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

7.13.2 **NonEmptyOctetString** 类型以及 **one-meg** 值在 7.2.1 节定义。

7.13.3 **IdentifyingStringOrIndex** 类型表示一个携带了标识信息的字符串。

注 — 这类字符串的示例为元素或属性的前缀、命名空间名字和本地名字等。

7.13.4 该 ASN.1 类型的某个抽象值，或者具有一个（给定类别的）字符串作为 **NonEmptyOctetString** 类型的一个值，或者具有一个词汇表索引，该词汇表索引是（给定类别的）字符串在一个被称为可应用串表中的针对此类别字符串的词汇表索引（见 8.4.2 节）。

注 1 — 只要此类型被使用，则串的类别总是在之前小节的相关文本中定义（见7.5到7.12节）。

注 2 — "标识的（Identifying）"字符串与"未标识的（non-identifying）" 字符串的处理是不同的（见7.14节）。一个未标识的字符串可能以多种编码格式中的一种来进行编码，而所有标识的字符串都以UTF-8来进行编码。另外，一个未标识的字符串可能或不可能被加入到动态串表中（由创建者来选择）（见7.14.6节），而标识的字符串总是能够被加入到动态串表中（见7.13.7节）。

7.13.5 **`literal-character-string`**，如果存在的话，将携带字符串（见 7.13.4 节）的 UTF-8 编码（见 ISO/IEC 10646，附件 D）。

7.13.6 **`string-index`**，如果存在的话，将包含可应用串表中的与该字符串相同的任意条目的词汇表索引。

7.13.7 在创建此 ASN.1 类型的一个抽象值时（表示一个给定类别的给定字符串），如果在可应用串表的当前内容中存在一个相同的字符串，则创建者应当执行下面的动作 a)或动作 b)作为一种实现时的选项（但是如果可能的话还是应当首选第一选项，因为它产生的指向相同字符串的索引的数量是最少的），否则（如果不存在相同的字符串），则创建者应当执行下面的动作 b）。动作 a)和动作 b)定义如下：

a） 选择**`string-index`**可选项，并且为**`string-index`** 分配与该字符串相同的任意现存条目的词汇表索引；

b） 选择**`literal-character-string`**可选项，为**`literal-character-string`**分配给定的字符串，并将一个相同的字符串加入到可应用串表中，除非该表已经包含了2^{20} 个条目。

注 — 选择执行动作b) 将会导致在串表中出现多个相同的字符串（如果该表尚未包含2^{20} 个条目）。但这不会影响对字符串的后续处理（见7.13.8节）。

7.13.8 在处理一个表示（给定类别的）字符串的 ASN.1 类型的抽象值时，处理者应当按照如下方式来决定此抽象值所表示的字符串：

a） 如果**`string-index`** 可选项存在，则由抽象值所表示的字符串应当是可应用串表当前内容中的字符串，其词汇表索引是**`string-index`**的值。

b） 如果**`literal-character-string`** 可选项存在，则由此抽象值所表示的字符串应当是**`literal-character-string`**的值，且一个相同的字符串应当被加入到此可应用串表中（见7.13.9节），除非该表已经包含了2^{20} 个条目。

注 — 选择执行动作b) 将会导致在串表中出现多个相同的字符串（如果该表尚未包含2^{20} 个条目）。但这不会影响对字符串的后续处理（见7.13.8节）。

7.13.9 如果一个处理者不能向某个尚未包含 2^{20} 个条目的词汇表中增加字符串（由于任意原因，包括实现特定的限制），而这种增加是 7.13.8 b)中所要求的，则应当停止处理此快速信息集文档，并发出一个错误。

7.14 `NonIdentifyingStringOrIndex` 类型

7.14.1 **`NonIdentifyingStringOrIndex`** 类型的定义为：

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

7.14.2 **`EncodedCharacterString`** 类型和 **`one-meg`** 的值分别在 7.17 节和 7.2.1 节定义。

7.14.3 **`NonIdentifyingStringOrIndex`** 类型表示一个未携带标识信息的字符串。

注 — 这种字符串的示例为某个属性的值。

7.14.4 **`NonIdentifyingStringOrIndex`** 类型的某个抽象值，或者具有一个（给定类别的）字符串作为 **`EncodedCharacterString`** 类型的值（见 7.17 节），或者具有一个词汇表索引，该词汇表索引是（给定类别的）字符串在一个被称为可应用串表中的针对此类别字符串的词汇表索引（见 8.4.2 节）。

注 — 只要此类型被使用，则串的类别总是在之前小节的相关文本中定义。

7.14.5 **`string-index`**，如果存在的话，则取值或者是 0（表示一个 0 长度的字符串，见 7.14.6 节），或者是应当包含可应用串表中的与该字符串相同的任意条目的词汇表索引。

7.14.6 对于一个 0 长度的字符串，创建者将总是使用 **string-index** 可选项，且取整数值为 0。一个处理者应当认为该值表示一个 0 长度的字符串。

7.14.7 在创建此 **NonIdentifyingStringOrIndex** 类型的一个非 0 长度的抽象值时（表示一个给定类别的给定字符串），如果在可应用串表的当前内容中存在一个相同的字符串，则创建者应当执行下面的动作 a)或动作 b)作为一种实现时的选项（但是如果可能的话还是应当首选第一选项，因为它产生的指向相同字符串的索引的数量是最少的），否则（如果不存在相同的字符串），则创建者应当执行下面的动作 b）。动作 a)和动作 b)定义如下：

a） 选择**string-index**可选项，并且为**string-index** 分配与该字符串相同的任意现存条目的词汇表索引；

b） 选择**literal-character-string**可选项，并且为**character-string**组件分配给定的字符串，并且或者：

1） 将一个相同的字符串加入到可应用串表中，并设置**add-to-table**组件的值为**TRUE**（如果该可应用串表已经包含了2^{20} 个条目，则不能使用此动作b1）；或者

注 1 — 如果可应用串表已经包含了 2^{20} 个条目，则只有动作 a 或动作 b2 是可用的。

2） 设置**add-to-table** 组件的值为**FALSE**。

注 2 — 选择执行动作b1将会导致在当前内容中出现多个相同的字符串。但这不会影响对字符串的后续处理（见7.14.8节）。

7.14.8 在处理一个表示给定类别的字符串的 ASN.1 类型的抽象值时，处理者应当按照如下方式来决定此抽象值所表示的字符串：

a） 如果**string-index**可选项存在，则由该抽象值所表示的字符串应当是可应用串表中的字符串，其词汇表索引是**string-index**的值。

注 1 — 如果 **string-index** 超出了该词汇表的当前长度，则快速信息集文档出错。

b） 如果**literal-character-string** 可选项存在，且**add-to-table**组件的值为**TRUE**，则由该抽象值所表示的字符串应当是**character-string**组件的值。处理者应当向此可应用串表中增加一个相同的字符串（见7.14.12节）。

注 2 — 如果可应用串表已经包含了 2^{20} 个串，则快速信息集文档出错。

c） 如果**literal-character-string** 可选项存在，且**add-to-table**组件的值为**FALSE**，则由该抽象值所表示的字符串应当是**character-string**组件的值。

7.14.9 如果一个处理者不能向某个词汇表中增加字符串（由于任意原因，包括实现特定的限制），而这种增加是7.14.8 b)中所要求的，则应当停止处理此快速信息集文档，并应当发出一个错误。

7.15 NameSurrogate 类型

7.15.1 **NameSurrogate** 类型的定义为：

```
NameSurrogate ::= SEQUENCE {
     prefix-string-index           INTEGER(1..one-meg) OPTIONAL,
     namespace-name-string-index   INTEGER(1..one-meg) OPTIONAL,
     local-name-string-index       INTEGER(1..one-meg) }
     (CONSTRAINED BY  {--当且仅当 namespace-name-string-index 存在时,
                      -- prefix-string-index 才存在 --})
```

7.15.2 **one-meg** 的值在 7.2.1 节中定义。

7.15.3 **NameSurrogate** 类型具有三个正整数（前两个是可选的），并由此构成一个名字代理（见 8.5.2 节）。

注 — 此类型仅出现在**Document**类型的**initial-vocabulary** 组件中。

7.15.4 **prefix-string-index**（如果存在的话），**namespace-name-string-index** （如果存在的话）和**local-name-string-index** 应当大于 0，且分别不大于初始词汇中 PREFIX 表，NAMESPACE NAME 表和 LOCAL-NAME 表中条目的个数。

注 — 如果一个快速信息集文档的处理者选择不对是否满足此限制进行检查，则可能会在进一步的处理中出现安全弱点。

7.15.5 **prefix-string-index** 应当缺失，除非 **namespace-name-string-index** 存在。

7.16 QualifiedNameOrIndex 类型

7.16.1 **QualifiedNameOrIndex** 类型的定义为：

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

7.16.2 **IdentifyingStringOrIndex** 类型和 **one-meg** 的值分别在 7.13 节和 7.2.1 节定义。

7.16.3 **QualifiedNameOrIndex** 类型表示一个限定名字（见 3.4.11）。

7.16.4 该 ASN.1 类型的一个抽象值或者具有三个组件，分别对应一个（给定类别的）限定名字的前缀、命名空间名字和本地名字，或者具有一个词汇表索引，该词汇表索引是一个给定类别的名字代理在一个被称为可应用名字表中针对此限定名字类别的词汇表索引（见 8.5.2 节）。

注 — 只要此类型被使用，则类别总是在之前小节的相关文本中定义。

7.16.5 **name-surrogate-index**，如果存在的话，应当包含一个名字代理在可应用名字表中的词汇表索引。

7.16.6 如果 **namespace-name** 缺失，则 **prefix** 也应当缺失。

7.16.7 在创建一个表示（给定类别的）给定限定名字的 ASN.1 类型的抽象值时，创建者应当按照下面小节规定的方式来执行动作。

7.16.7.1 应当按照顺序对下列条件进行判定：

a） 或者限定名字没有前缀，或者限定名字的前缀存在于PREFIX表的当前内容中；

b） 或者限定名字没有命名空间名字，或者限定名字的命名空间名字存在于NAMESPACE NAME表的当前内容中；

c） 限定名字的本地名字存在于LOCAL NAME表的当前内容中；

d） 如果前三个条件都满足，且有一个包含前缀（如果有的话）、命名空间名字（如果有的话）和本地名字的名字代理（见8.5节）存在于可应用名字表的当前内容中。

7.16.7.2 如果上面的所有条件都满足，则应当选择 **name-surrogate-index** 选择项，并且应当被赋值为 7.16.7.1 d）所确定的名字代理在可应用名字表中的词汇表索引，然后完成 7.16.7 节的过程。

7.16.7.3 否则，应当选择 **literal-qualified-name** 选择项，其组件应当按照如下所述来赋值：

a） 如果限定名字没有前缀，则**prefix** 组件应当缺失，否则应当通过应用7.13.7节将前缀赋值给**prefix**组件，并有一个限制，即如果有一个相同的字符串存在于可应用串表的当前内容中，则不应当执行7.13.7 b)的动作。该组件的类型为**IdentifyingStringOrIndex** （见7.13节），在这里表示一个PREFIX 类别的字符串；

b） 如果限定名字没有命名空间名字，则**namespace-name** 组件应当缺失，否则应当通过应用7.13.7节将命名空间名字赋值给**namespace-name** 组件，并有一个限制，即如果有一个相同的字符串存在于可应用串表的当前内容中，则不应当执行7.13.7 b) 的动作。该组件的类型为**IdentifyingStringOrIndex** （见7.13节），在这里表示一个NAMESPACE NAME 类别的字符串（见8.4.2节）；

c） 限定名字的本地名字应当（通过应用7.13.7节）被赋值给**local-name**组件。该组件的类型为**IdentifyingStringOrIndex**（见7.13节），在这里表示一个LOCAL NAME 类别的字符串。

注 — 本小节中应用 7.13.7 节，可能会导致将本地名字加入到 LOCAL NAME 表中。

7.16.7.4 如果 7.16.7.3 节中，在上述三个字符串中的其中一个或多个字符串上应用 7.13.7 节的内容，但由于表已经包含了 2^{20} 个条目，而没有将相应字符串加入到词汇表中（见 7.13.7 b），则不能为此限定名字创建一个名字代理。

7.16.7.5 否则，应当创建一个包含前缀（如果有的话）词汇表索引、命名空间名字（如果有的话）词汇表索引和本地名字词汇表索引的名字代理。如果此名字代理不存在于可应用名字表的当前内容中，则它应当被加入到此表中，除非该表已经包含了 2^{20} 个条目。

7.16.8 在处理一个表示给定类别的限定名字的 **QualifiedNameOrIndex** 类型的抽象值时，处理者应当按照下面小节的规定来判断由此抽象值所表示的限定名字。

7.16.8.1 如果 **name-surrogate-index** 选择项存在，则由抽象值所表示的限定名字应当是可应用名字表中给定类别的名字代理所表示的限定名字（见 8.5.2 节），而且该名字代理的词汇表索引是 **name-surrogate-index** 的值。

7.16.8.2 如果 **literal-qualified-name** 选择项存在，则

a） 抽象值所表示的限定名字应当根据如下所述来判断：

1） 限定名字的前缀应当（通过应用7.13.8节）根据**prefix**组件（如果存在的话）来判断；该组件的类型为**IdentifyingStringOrIndex**（见7.13节），在这里表示一个PREFIX类别的字符串；

2） 限定名字的命名空间名字应当（通过应用7.13.8）根据**namespace-name**组件（如果存在的话）来判断；该组件的类型为**IdentifyingStringOrIndex**（见7.13节），在这里表示一个NAMESPACE NAME类别的字符串（见8.4.2节）；

3） 限定名字的本地名字应当（通过应用7.13.8节）根据**local-name**组件来判断；该组件的类型为**IdentifyingStringOrIndex**（见7.13节），在这里表示一个LOCAL NAME类别的字符串。

注 — 这些动作可能会要求加入到相应的词汇表中，如7.13.8 b中的规定。

b） 在处理**literal-qualified-name**的组件并进行了可能的加入后，如果词汇表索引对存在的所有组件都可用，则一个包含了那些词汇表索引的名字代理应当被加入到可应用名字表中（但见7.16.9节），除非该词汇表已经包含了2^{20} 名字代理。

7.16.9 如果一个处理者不能向某个包含的条目少于 2^{20} 个的词汇表中增加名字代理（由于任意原因，包括实现特定的限制），而这种增加是 7.16.8.2 节 b)中所要求的，则应当停止处理此快速信息集文档，并应当发出一个错误。

7.17 EncodedCharacterString 类型

7.17.1 **EncodedCharacterString** 类型的定义为：

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

7.17.2 **EncodedCharacterString** 类型包含了一个字符串的编码。它规定了一个八位组串，该八位组串与字符串之间可以进行可逆映射。一个快速信息集文档的创建者在 **encoding-format** 中规定了正在使用的编码，而处理者使用此信息将 **octets** 组件中的八位组解码为一个字符串。

7.17.3 组件 **octets** 应当携带一个八位组串值，该值是 **encoding-format** 组件中规定的字符串的编码。

注 — 一般来说，对于一个给定的字符串，可以使用多种编码。一个快速信息集文档的创建者可能会基于某种标准来选择一种编码（例如，可以尽力优化长度或优化处理速度），他也可能会倾向于使用UTF-8或UTF-16BE的便利性和全球适用性。还有这些一种情况，即某些编码将仅能够对那些包含ISO/IEC 10646字符的某个子集的字符串进行编码。

7.17.4 **`utf-8`** 编码格式可应用于任何字符串。此编码格式的使用可以通过为字符串产生一个 UTF-8 编码（见 ISO/IEC 10646），并将该编码赋值给 **`octets`** 组件而实现。

注 — 这种编码格式对下述字符串是最适宜的，即ISO/IEC 10646基本多语言平面前面部分的ISO/IEC 10646字符在字符串中是最常用的，且没有应用其他编码格式或没有其他编码格式是更适用的。

7.17.5 **`utf-16`** 编码格式也可应用于任何字符串。此编码格式的使用可以通过为字符串产生一个 UTF-16BE 编码（见 Unicode，2.6），并将该编码赋值为 **`octets`** 组件而实现。

注 1 — 这种编码格式对下述字符串是最适宜的，即在字符串中使用了更广泛的ISO/IEC 10646字符，且没有应用其他编码格式或没有其他编码格式是更适用的。

注 2 — Unicode的2.6中规定的UTF-16BE编码的字节顺序是最高位字节在前（即处于八位组串的前面字节）。

7.17.6 **`restricted-alphabet`** 编码格式基于一个有限字母表的使用，此有限字母表是从某个有限字母表中存在的字母中选择而来的。**`restricted-alphabet`** 组件应当包含此有限字母表的词汇表索引。这种编码格式仅能应用于下述字符串，即完全包含有限字母表中的字符，而此有限字母表是在 **`restricted-alphabet`** 组件所指向的有限字母表条目中。一个 **`restricted-alphabet`** 编码格式应当根据 7.17.6.1 节到 7.17.6.6 节的规定来应用。

7.17.6.1 应当为有限字母表中的每个字符分配一个整数值（起始为 0），并且按顺序分配。

7.17.6.2 字符串中的每个字符都应当被转换为一个整数，该整数是分配给有限字母表中的字符的。

7.17.6.3 在一个比特字段中，每个整数值都应当被表示为一个无符号整数的二进制数。比特字段的长度由分配给有限字母表中的最后一个字符的整数值来决定。将该整数值再加上 1 来产生一个整数值（即 N）。比特字段的长度应当是能够将 N 编码为一个无符号整数编码的最少的比特数。

注 1 — 加1是必要的，因为在比特字段中，一个全1的值被解释为是字符串的结束，而不能用来表示一个字符。这就意味着如果有限字母表包含的字符个数正好是2的幂次，则比特字段将比所期望的多一个比特。

注 2 — 例如，如果在有限字母表中有24个字符，则每个字符将被编码为5个比特，但是如果在有限字母表中有32个字符，则每个字符将被编码为6个比特。

7.17.6.4 所有这些比特字段应当被级联起来（按顺序）形成一个比特串。

7.17.6.5 如果由此形成的比特串的长度不是 8 比特的整数倍，则应当在后面附加‘1’使得比特串的长度成为 8 比特的整数倍。

7.17.6.6 由此形成的比特串（此时是 8 比特的整数倍），被解释为一个八位组串，应当赋值给 **`octets`** 组件。

7.17.7 **`encoding-algorithm`** 编码格式由编码算法规定（见 8.3 节），此算法为编码算法表中的表项，其词汇表索引是 **`encoding-algorithm`** 组件的值。编码算法词汇表索引应当赋值给 **`encoding-algorithm`** 组件，且由该编码所生成的八位组串应当被置于 **`octets`** 组件中。

8 快速信息集文档的构造和处理

一个快速信息集文档在各种表中都大量使用了词汇表索引，这些表是在该文档的构建和处理的不同阶段中构造的。8.1 小节规定了 **`Document`** 类型的某个抽象值中组件的概念顺序，以便确保快速信息集文档的创建者和处理者能够创建相同的词汇表。后续的小节规定了在创建和处理一个快速信息集文档的过程中构造和使用的那些表。这些表在计算机系统中的表示是实现时考虑的问题，不作为标准化的内容。一个词汇表提供了词汇表索引到一个 XML 信息集之间的映射（可能是非直接的）。

注 — 某个快速信息集文档的词汇表是在构造此快速信息集文档的过程中动态构造的。在快速信息集文档的处理过程中，它们根据快速信息集文档的内容而动态构造。它们不会以任何其他形式进行交互。

8.1 Document类型的抽象值中组件的概念顺序

8.1.1 为了确保在构造和处理一个快速信息集文档时，不同的实现能够以相同的方式分配词汇表索引，因此对 **Document** 类型的抽象值中组件的概念顺序作出了规定。在构造和处理这样的抽象值时，如果要将串（见 7.13.7 节和 7.14.6 节）和名字代理（见 7.16.7 节）加入到词汇表中，则应当使用这种概念顺序。

注 — 此顺序与对快速信息集文档的组件进行编码的顺序是相同的。它并没有一定意味着文档所携带的语义也是照此顺序进行处理。定义此顺序的目的仅仅是为了确保一个快速信息集文档的创建者和处理者对任意一个给定的词汇表条目都能够分配相同的词汇表索引。

8.1.2 构造和处理一个快速信息集文档的概念顺序定义如下：访问 **Document** 类型的某个抽象值中的所有组件，应当根据 8.1.2.1 节到 8.1.2.5 节所规定的算法进行访问。组件被访问的顺序便定义了概念顺序。

8.1.2.1 抽象值的顶层组件（相应于 **Document** 类型）应当被首先访问。

8.1.2.2 如果正在被访问的组件的类型是一个顺序（sequence）类型，则出现在抽象值中的顺序类型中的组件应当按照它们的文本定义的顺序来访问，顺序为从出现的第一个组件开始到出现的最后一个组件结束，然后为每个被访问的组件递归应用 8.1.2.1 节到 8.1.2.5 节。

8.1.2.3 如果正在被访问的组件的类型是一个数组（sequence-of）类型，则出现在 sequence-of 中的组件应当按照 sequence-of 的顺序来访问，顺序为 sequence-of 中出现的第一个组件开始到出现的最后一个组件结束，然后为每个被访问的组件递归应用 8.1.2.1 节到 8.1.2.5 节。

8.1.2.4 如果正在被访问的组件的类型是一个选择（choice）类型，则应当访问出现在抽象值中的选择项，并对此选择项递归应用 8.1.2.1 节到 8.1.2.5 节。

8.1.2.5 如果正在被访问的组件的类型是任何其他的 ASN.1 类型，则对此组件不需要做任何更进一步的动作。

8.2 有限字母表

8.2.1 每个快速信息集文档都有一个与之相关联的有限字母表。有限字母表中包含有限字母，这些字母可以通过一个词汇表索引被引用。

8.2.2 处于有限字母表中的每个条目应当是不同的 ISO/IEC 10646 字符组成的一个有序集合，集合的长度可以任意，在 2 和 2^{20} 个字符之间。

注 — 一个有限字母允许对完全包含了该集合中所有字符的一个字符串进行压缩编码，方法是为集合中的每个字符分配一个累进的整数值，然后使用这些整数来对串中的字符进行编码（见7.17.6节）。

8.3 编码算法表

8.3.1 每个快速信息集文档都有一个与之相关联的编码算法表。编码算法表中包含有编码算法的定义，这些算法可以通过一个词汇表索引被引用。

8.3.2 此表中的每个条目都根据某些定义的特性规定了如何将一个字符串编码为一个八位组串（见 7.17.7 节）。

注 — 所定义的特性可能会涉及到串的长度，出现在其内的字符，或者字符的某个任意复杂的模式等。一般来说，一个给定的编码算法仅应用于ISO/IEC 10646字符串的一个特殊的已定义子集。

8.3.3 编码算法受到如下所述的限制：

a） 编码算法应当具有一个与之相关联的URI，除非它是一个内置的编码算法，因此它在被加入到表中时可以被引用到；

b） 编码算法应当精确地规定它可应用于什么类型的字符串；这种规定应当包括一个有限字母表（如果有的话），一个长度范围（如果有的话），以及对字符串所施加的关于长度和内容方面的任何附加限制（如模式）；

c） 对于可以应用该编码算法的任意字符串，编码算法应当提供一个可逆的从字符串到八位组串的映射。

注 1 — 如上所述就隐含了不会存在这样的一个字符串S，执行编码步骤后由S变为E，然后再解码由E变为S'，而导致S' ≠ S，即使S和S'之间的差异是非常小的（例如，存在一个多余的空格）。但另一方面，它并不要求所有的字符串S都是可编码的，也不要求所有的编码都是规范的。

注 2 — 根据内存中的数据，如浮点数，而创建一个快速信息集文档的应用并不要求为此数据产生一种词法表示，然后再对此表示应用一种编码算法。替代的是，该应用可以直接根据此数据创建一个八位组串，假若该八位组串是可以通过对一个字符串应用此编码算法而产生的，而该字符串是表示了此内存数据，并且是可以应用此编码算法的一个字符串。

注 3 — 编码算法（不是内置编码算法）可能在其他标准中规定，或者在一个快速信息集文档的创建者和处理者之间进行互相协商。

8.4 动态串表

8.4.1 每个快速信息集文档都有八个与之相关联的动态串表。每个动态串表都包含了字符串，这些字符串可以通过一个词汇表索引被引用。

8.4.2 本建议书 | 国际标准将能够出现在一个快速信息集文档中的所有字符串分类为下面的八种类别，每种类别都有一个动态串表：

a） PREFIX：此类别所包含的字符串是作为一个**element**信息项，**attribute**信息项或**namespace**信息项中的**[prefix]** 属性的那些字符串。

b） NAMESPACE NAME：此类别所包含的字符串是作为一个**element**信息项，**attribute**信息项或**namespace**信息项中的**[namespace name]**属性的那些字符串。

c） LOCAL NAME：此类别所包含的字符串是作为一个**element**信息项或**attribute**信息项中的**[local name]**属性的那些字符串。

d） OTHER NCNAME：此类别所包含的字符串是作为一个**processing instruction**信息项中的**[target]**属性的那些字符串；或者是作为一个**unexpanded entity reference**信息项，**unparsed entity**信息项或**notation**信息项中的**[name]**属性的那些字符串；或者是作为一个**unparsed entity**信息项中的**[notation name]**属性的那些字符串。

e） OTHER URI：此类别所包含的字符串是作为一个**unexpanded entity reference**信息项，**document type declaration**信息项，**unparsed entity**信息项或**notation**信息项中的**[system identifier]**属性或**[public identifier]**属性的那些字符串。

f） ATTRIBUTE VALUE：此类别所包含的字符串是作为一个**attribute**信息项中的**[normalized value]**属性的那些字符串。

g） CONTENT CHARACTER CHUNK：此类别所包含的字符串是作为一个**character**信息项集合中的**[character code]**属性的那些字符串，而此信息项的集合是一个给定的**element**信息项的连续子项。

h） OTHER STRING：此类别所包含的字符串是作为一个**document**信息项中的**[version]**属性的那些字符串；或者是作为一个**processing instruction**或**comment**信息项中的**[content]**属性的那些字符串。

8.5 动态名字表和名字代理

8.5.1 每个快速信息集文档都有两个与之相关联的动态名字表。每个动态名字表都包含名字代理，这些名字代理可以通过一个词汇表索引被引用，并且被用于标识一个限定名字，限定名字可能或者有前缀，或者没有前缀（同时可能有一个命名空间名字，也可能没有）。

8.5.2 一个名字代理是最多三个有序的词汇表索引的集合：

a）（可选的）一个串在PREFIX表中的索引；

b）（可选的）一个串在NAMESPACE NAME表中的索引，以及

c） 一个串在LOCAL NAME表中的索引。

第一个词汇表索引不应当存在，除非第二个存在。

8.5.3 有三种情况会发生：

a） 所有三个索引都存在，在这种情况下，此名字代理表示一个带前缀的限定名字；

b） 仅有第二个和第三个索引存在，在这种情况下，此名字代理表示一个不带前缀，但具有一个命名空间名字的限定名字；

c） 仅有第三个索引存在，在这种情况下，此名字代理表示一个不带前缀且没有命名空间名字的限定名字。

8.5.4 本建议书 | 国际标准对一个快速信息集文档中可能出现的所有限定名字进行了分类（因此也对表示它们的名字代理进行了分类），将其分为下面的两种类别，每种都有一个动态名字表：

a） ELEMENT NAME：此类别所包含的名字代理是表示一个**element**信息项的限定名字的那些名字代理。

b） ATTRIBUTE NAME：此类别所包含的名字代理是表示一个**attribute**信息项的限定名字的那些名字代理。

8.5.5 由一个给定名字代理所表示的限定名字应当根据如下所述来判断，给定一个动态串表：

a） 第一个词汇表索引（如果有的话）应当被解释为一个字符串在PREFIX 表中的词汇表索引；以及

b） 名字代理的第二个词汇表索引（如果有的话）应当被解释为一个字符串在NAMESPACE NAME表中的词汇表索引；以及

c） 第三个整数应当被解释为一个字符串在LOCAL NAME表中的词汇表索引。

9 内置的有限字母

9.1 "数字"有限字母

9.1.1 该有限字母的词汇表索引为 1，并包括如下 15 种 ISO/IEC 10646 字符（按此顺序）：

数字 0 到数字 9
连字符－减号
加号
句号
拉丁文的小写字母 E
空格

9.1.2 该有限字母适用于对表示多种数字类型的字符串进行编码，包括使用科学计数法的浮点数。一个单独的字符串可能会包含多个以空格分隔的数字。

9.2 "日期和时间"有限字母

9.2.1 该有限字母的词汇表索引为 2，并包括如下 15 种 ISO/IEC 10646 字符（按此顺序）：

数字 0 到数字 9
连字符－减号
冒号
拉丁文的大写字母 T
拉丁文的大写字母 Z
空格

9.2.2 该有限字母适用于对基于 ISO 8601 的日期和时间的最常用表达式进行表示的字符串进行编码。一个单独的字符串可能会包含多个以空格分隔的这种表达式。

10 内置的编码算法

10.1 概要

10.1.1 本节规定了内置的编码算法。内置编码算法没有与之相关联的 URI。

注 — URI对于某个初始词汇表中显式标识的编码算法是必要的，但是内置的编码算法总是被隐含加入到编码算法表中的，因此没有必要使用URI。

10.1.2 在本节中，术语"单词"表示一个给定字符串中的任意连续字符的组合，其中：

a） 不包括空格；且

b） 或者处于字符串的起始位置，或者在之前有一个空格；且

c） 或者处于字符串的结尾位置，或者在其后有一个空格。

注 — 一个"单词"不限定在字母表中的字符。

10.2 "十六进制"编码算法

10.2.1 该编码算法的词汇表索引为 1，并且仅能够应用于包含下述 16 种 ISO/IEC 10646 字符的字符串：

数字 0 到数字 9

拉丁文的大写字母 A 到拉丁文的大写字母 F

且该字符串所包含的字符为偶数个（包括 0 个）。

注 — 不允许内置的XML空白字符。

10.2.2 该字符串应当被解释为一个八位组串的十六进制编码，其中串的第一个字符对应于第一个八位组的最高位半元组，依此类推。

10.3 "base64"编码算法

10.3.1 该编码算法的词汇表索引为 2，并且仅能够应用于下列字符串：

a） 包括的全部是拉丁文大写字母A到拉丁文大写字母Z，拉丁文小写字母a到拉丁文小写字母z，数字0到数字9，加号，斜线，以及等于号等字符；并且

注 — 不允许在字符串中出现 XML 空白字符。

b） 或者是根据IETF RFC 2045的6.8节规定的“内容传送编码”的一个有效实例；或者是根据IETF RFC 2045的要求，在需要时插入XML空白字符后变成为该编码的一个有效实例。

10.3.2 该字符串应当被解释为一个八位组串（假设根据 IETF RFC 2045 的需求，在需要时出现了所需的附加的 XML 空白字符）的 Base64 编码（见 IETF RFC 2045）。作为结果的八位组串是根据此 Base64 编码所规定的八位组串。

10.3.3 该编码算法适用于编码的字符串是不包含 XML 空白字符，且为（任意长度的）Base64 串的字符串，或者是通过添加 XML 空白字符而变为 Base64 串的字符串。

10.4 "短整数"编码算法

10.4.1 该编码算法的词汇表索引为 3，并且仅能够应用于满足下列条件的字符串：

a） 包括的全部是数字0到数字9，连字符-减号，以及空格等字符；并且

b） 字符串中的第一个字符和最后一个字符都不是一个空格，且没有两个连续的空格出现；且

c） 字符串至少包含一个单词（见10.1.2节）；且

d） 每个出现的连字符-减号都是单词中的第一个字符；且

e） 每个出现的连字符-减号的后面都至少跟一个数字0到数字9范围内的字符；且

f） 每个数字0或者是单词中的唯一一个字符，或者在其前面有一个数字0到数字9范围内的字符；且

g） 字符串中的每个单词，如果被解释为是一个有符号整数的十进制字符串，则其值的范围应限制在–32768到32767之间。

10.4.2 字符串中的每个单词（见 10.1.2 节）都应当被解释为是一个有符号整数的十进制字符串，并被表示为一个 16 比特的两个互补整数。

10.4.3 在代表一个单词的 16 比特的两个互补整数中，每个 8 比特组都应当产生八位组串中的一个八位组，其中起始为最高阶的 8 比特组。在每个 8 比特组中的最高阶比特应当是相应八位组中的最高有效位。如果在字符串中有多个单词，则应当对它们按顺序进行编码，并且由多个 16 比特的两个互补整数所产生的八位组应当按此顺序级联起来。

10.4.4 该编码算法适用于编码的字符串是一个表示范围在–32768 到 32767 之间的整数（可表示为一个 16 比特的两个互补整数）的字符串，或者是表示此种整数列表的一个字符串。

10.5 "整数"编码算法

10.5.1 该编码算法的词汇表索引为 4，并且仅能够应用于满足下列条件的字符串：

a） 字符串满足10.4.1中a 到f 所规定的条件；且

b） 字符串中的每个单词（见10.1.2节），如果被解释为是一个有符号整数的十进制字符串，则其值的范围应限制在–2147483648到2147483647之间。

10.5.2 字符串中的每个单词（见 10.1.2 节）都应当被解释为一个有符号整数的十进制字符串，并被表示为一个 32 比特的两个互补整数。

10.5.3 在代表一个单词的 32 比特的两个互补整数中，每个 8 比特组都应当产生八位组串中的一个八位组，其中起始为最高阶的 8 比特组。在每个 8 比特组中的最高阶比特应当是相应八位组中的最高有效位。如果在字符串中有多个单词，则应当对它们按顺序进行编码，并且由多个 32 比特的两个互补整数所产生的八位组应当按此顺序级联起来。

10.5.4 该编码算法适用于编码的字符串是一个表示范围在–2147483648 到 2147483647 之间的整数（可表示为一个 32 比特的两个互补整数）的字符串，或者是表示此种整数列表的一个字符串。

10.6 "长整数"编码算法

10.6.1 该编码算法的词汇表索引为 5，并且仅能够应用于满足下列条件的字符串：

a） 字符串满足10.4.1中a 到f 所规定的条件；且

b） 字符串中的每个单词（见10.1.2节），如果被解释为是一个有符号整数的十进制字符串，则其值的范围应限制在–9223372036854775808到9223372036854775807之间。

10.6.2 字符串中的每个单词（见 10.1.2 节）都应当被解释为一个有符号整数的十进制字符串，并被表示为一个 64 比特的两个互补整数。

10.6.3 在代表一个单词的 64 比特的两个互补整数中，每个 8 比特组都应当产生八位组串中的一个八位组，其中起始为最高阶的 8 比特组。在每个 8 比特组中的最高阶比特应当是相应八位组中的最高有效位。如果在字符串中有多个单词，则应当对它们按顺序进行编码，并且由多个 64 比特的两个互补整数所产生的八位组应当按此顺序级联起来。

10.6.4 该编码算法适用于编码的字符串是一个表示范围在–9223372036854775808 到 9223372036854775807 之间的整数（可表示为一个 64 比特的两个互补整数）的字符串，或者是表示此种整数列表的一个字符串。

10.7 "布尔型"编码算法

10.7.1 该编码算法的词汇表索引为 6，并且仅能够应用于满足下列所有条件的字符串：

a） 字符串全部是由一个或多个单词"false" 或"true"，以及空格字符构成的；且

b） 字符串中的第一个字符和最后一个字符都不是一个空格，且没有两个连续的空格出现；且

c） 字符串至少包含一个单词（见10.1.2节）。

10.7.2 字符串中的每个单词"false"或"true"应当被编码为所生成的八位组中的一个单独比特（分别设置为 0 或 1），起始为第一个八位组的第 5 个比特一直到第 8 个比特。后续的比特被置于后续的八位组中，从每个八位组的第一个比特开始一直到第 8 个比特，需要用到多少个八位组就用多少个。在最后一个八位组中未使用的比特都被设置为 0。

10.7.3 第一个八位组中的前四个比特应当包含最后一个八位组中未使用比特的个数，并将其编码为一个 4 比特的无符号整数。

注 — 第一个八位组有可能也是最后一个八位组，并可能包含最多3个未用比特。如果有多个八位组，则最后一个比特可能包含最多7个未用比特。

10.8 "浮点数"编码算法

10.8.1 该编码算法的词汇表索引为 7，并且仅能够应用于满足下列所有条件的字符串：

a） 字符串包含的全部是数字0到数字9，连字符-减号，句号，拉丁文大写字母E，以及空格等字符；并且
注 — 不允许使用拉丁小写字母 e，因此编码是不可逆的

b） 字符串中的第一个字符和最后一个字符都不是一个空格，且没有两个连续的空格出现；且

c） 字符串至少包含一个单词（见10.1.2节）；且

d） 字符串中的每个单词都与W3C XML模式第2部分3.2.4节中规定的浮点数的规范词法表示相匹配；且

e） 字符串中的每个单词，如果被解释为是表示一个浮点十进制数字的字符串，则将产生一个可表示为一个32比特的IEEE 754浮点数。

10.8.2 字符串中的每个单词（见 10.1.2 节）都应当被解释为一个浮点十进制数字字符串，并被表示为一个 32 比特的 IEEE 754 浮点数。

10.8.3 在代表一个单词的 32 比特的 IEEE 754 浮点数中， 每个 8 比特组都应当产生八位组串中的一个八位组，其中起始为前导的 8 比特组。在每个 8 比特组中的前导比特应当成为相应八位组中的最高有效位。如果在字符串中有多个单词，则应当对它们按顺序进行编码，并且由多个 32 比特的 IEEE 754 浮点数所产生的八位组应当按此顺序级联起来。

10.8.4 该编码算法适用于编码的字符串是一个表示单个浮点数字（可表示为一个 32 比特的 IEEE 754 浮点数）的字符串，或者是表示此种浮点数字列表的一个字符串。

10.9 "双精度浮点数"编码算法

10.9.1 该编码算法的词汇表索引为 8，并且仅能够应用于满足下列所有条件的字符串：

a） 字符串满足10.8.1中a 到c 所规定的条件；且

b） 字符串中的每个单词都与W3C XML模式第2部分3.2.5节中规定的双精度浮点数的规范词法表示相匹配；且

c） 字符串中的每个单词（见10.1.2节），如果被解释为是表示一个浮点十进制数字的字符串，则将产生一个可表示为一个64比特的IEEE 754双精度浮点数。

10.9.2 字符串中的每个单词（见 10.1.2 节）都应当被解释为一个浮点十进制数字字符串，并被表示为一个 64 比特的 IEEE 754 双精度浮点数。

10.9.3 在代表一个单词的 64 比特的 IEEE 754 浮点数中， 每个 8 比特组都应当产生八位组串中的一个八位组，其中起始为前导的 8 比特组。在每个 8 比特组中的前导比特应当成为相应八位组中的最高有效位。如果在字符串中有多个单词，则应当对它们按顺序进行编码，并且由多个 64 比特的 IEEE 754 浮点数所产生的八位组应当按此顺序级联起来。

10.9.4 该编码算法适用于编码的字符串是一个表示单个浮点数字（可表示为一个 64 比特的 IEEE 754 双精度浮点数）的字符串，或者是表示此种浮点数字列表的一个字符串。

10.10 "uuid"编码算法

10.10.1 该编码算法的词汇表索引为 9，并且仅能够应用于满足下列所有条件的字符串：

a） 字符串包含的全部是数字0到数字9，拉丁小写字母A到拉丁小写字母F，连字符 — 减号，以及空格等字符；并且

b） 字符串中的第一个字符和最后一个字符都不是一个空格，且没有两个连续的空格出现；且

c） 字符串至少包含一个单词（见10.1.2节）；且

d） 每个单词都恰好包含36个字符；且

e） 在每个单词中，恰好有四个连字符 — 减号字符，并且所处的位置为第9、14、19和24（从1开始计数）。

10.10.2 字符串中的每个单词（见 10.1.2 节）应当被解释为一个 UUID（见 ITU-T X.667 建议书 | ISO/IEC 9834-8 中的 6.4 节)的十六进制表示，并且应当被表示为一个 16 个八位组的无符号整数，正如 ITU-T X.667 建议书 | ISO/IEC 9834-8 的 6.3 节所规定的那样。如果有多个单词，则多个 16 个八位组的无符号整数应当级联起来。

10.10.3 该编码算法适用于编码的字符串是一个表示单个 UUID 或 UUID 列表的字符串。

10.11 "cdata"编码算法

10.11.1 该编码算法的词汇表索引为 10，并且可被应用于任意字符串。

10.11.2 所产生的八位组串应当是字符串的 UTF-8 编码（见 ISO/IEC 10646）。

10.11.3 此算法仅能够被用于这样的 XML 信息集：即该 XML 信息集是通过对 XML 文档进行解析而创建的，并且附加的信息标识该字符串符合一个完整的 CDATA 部分（见 W3C XML 1.0 和 W3C XML 1.1）。如果在一个快速信息集文档中使用了此编码算法，则符合 CDATA 部分的所有字符串都应当应用此编码算法。

11 对所支持的XML信息集的限制及其他简化

11.1 本建议书 | 国际标准支持在实践中可能碰到的大部分 XML 信息集，但是不支持某些在理论上可能出现，而实践中一般不会出现的 XML 信息集。

11.2 本节中使用的术语“XML 自洽”具有下列含义：如果能够通过对某个适当的命名空间良构的 XML 文档进行解析后获得一个或多个信息项的一系列属性，则称该系列属性是“XML 自洽”的。

11.3 所支持的 XML 信息集符合下列所有条件：

a） **document**信息项的**[all declarations processed]** 属性取值为`true`；

b） 每个**element**信息项的**[in-scope namespaces]** 属性，以及所有**element**信息项的**[namespace attributes]**属性构成一个XML自洽的集合；

c） 每个**element**信息项的**[namespace name]**属性，以及所有**element**信息项的**[namespace attributes]**属性和该**element**信息项的**[prefix]** 属性构成一个XML自洽的集合；

d） 每个**attribute**信息项的**[namespace name]**属性，以及所有**element**信息项的**[namespace attributes]**属性和该**attribute**信息项的**[prefix]** 属性构成一个XML自洽的集合；

e） 每个**attribute**信息项的**[references]**属性，以及该**attribute**信息项的**[normalized value]** 属性构成一个XML自洽的集合；

f） 每个**processing instruction**信息项的**[notation]**属性，以及该**processing instruction**信息项的**[target]**属性和**document**信息项的**[notations]**属性构成一个XML自洽的集合；

g） 每个**unparsed entity**信息项的**[notation]**属性，以及该**unparsed entity**信息项的**[notation name]**属性和**document**信息项的**[notations]**属性构成一个XML自洽的集合；

h） 所有不表示空白字符的**character**信息项的**[element content whitespace]**属性的取值为**false**；

i） 每个**character**信息项的**[element content whitespace]**属性，以及该**character**信息项的**[character code]**属性构成一个XML自洽的集合；

j） 所有**attribute**信息项的**[normalized value]**属性以及所有**comment**信息项和**processing instruction**信息项的**[content]**属性都最多包含2^{32}个字符。

11.4 下列 XML 信息集中信息项的属性没有包含在表示这些信息项的 ASN.1 类型中：

a） **document**信息项的**[document element]**，**[base URI]**和**[all declarations processed]**属性（见7.2.30，7.2.31和7.2.32）；

b） **element**信息项的**[in-scope namespaces]**，**[base URI]**和**[parent]**属性（见7.3.8，7.3.9和7.3.10）；

c） **attribute**信息项的**[specified]**，**[attribute type]**，**[references]**和**[owner element]**属性（见7.4.7节，7.4.8节，7.4.9节和7.4.10节）；
d） **processing instruction**信息项的**[notation]**和**[parent]**属性（见7.5.7节和7.5.8节）；
e） **unexpanded entity reference**信息项的**[declaration base URI]**和**[parent]**属性（见7.6.7节和7.6.8节）；
f） **character**信息项的**[element content whitespace]**属性（见7.7.7节）；
g） **character**信息项的**[parent]**属性（见7.7.8节）；
h） **comment**信息项的**[parent]**属性（见7.8.6节）；
i） **document type declaration**信息项的**[parent]**属性（见7.9.7节）；
j） **unparsed entity**信息项的**[declaration base URI]**和**[notation]**属性（见7.10.8节和7.10.9节）；
k） **notation**信息项的**[declaration base URI]**属性（见7.11.7节）。

12 Document类型的比特级编码

12.1 本节规定了对 **Document** 类型的特殊编码，以便构成一个快速信息集文档。
注 — 设计这些特殊编码是为了优化处理和压缩速度，在本建议书 | 国际标准的许多预期使用中，这被认为是很关键的。

12.2 对编码的规定是通过一个编码者所执行的动作来完成的，结果是产生一些将被附加在一个比特流后面的比特。初始的比特流或者是空的，或者是包含了一个 XML 声明（见 12.3）。

12.3 一个 XML 声明（见 W3C XML 1.1 的 2.8 节）可能（作为创建者的一个选项）被包含在比特流的起始处。XML 声明（如果存在的话）应当是下列字符串之一，以 UTF-8 来编码：

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

12.4 XML 声明中的版本号（如果存在的话）应当被赋值为 **document** 信息项的相应**[version]**属性值。如果**[version]**属性无值，则 XML 声明中不应当包含版本号。

12.5 XML 声明中的单独声明（如果存在的话）应当被赋值为 **document** 信息项的相应**[standalone]**属性值。如果**[standalone]**属性无值，则 XML 声明中不应当包含单独声明。

12.6 然后，将 16 个比特'1110000000000000'附加在此比特流后面。
注 — 这些比特将或者出现在快速信息集文档的起始处，或者出现在XML声明的后面。在没有XML声明的情况下，词法解析器能够通过观察某个编码的前16个比特，来将一个潜在的快速信息集文档与任意良构的W3C XML 1.0或W3C XML 1.1文档区分开来，因为这16个比特永远都不会出现在一个良构的XML文档的起始处。

12.7 一个 16 个比特的比特字段，其中包含了本建议书 | 国际标准的版本号（见 12.9 节），并被编码为一个 16 比特的无符号整数，然后此比特字段应被附加在比特流之后。

12.8 然后，比特'0'（填充）应被附加在比特流之后。
注 — 这么做是为了确保在编码的后续部分字节能够对齐。

12.9 本建议书 | 国际标准这个版本的版本号为 1。
注 — 如果在新版本和之前版本中可能出现互操作问题时，希望本建议书 | 国际标准的后续版本增加版本号。

12.10 **`Document`** 类型的抽象值的 ECN 编码（见 ITU-T X.692 建议书 | ISO/IEC 8825-3），由 A.2 中的编码链接模块规定，应当被附加在比特流之后。

注 — 附件C为A.2中规定的编码提供了一个非正式的描述。

12.11 如果 **`Document`** 类型的抽象值的编码不是以一个八位组的最后一个比特来结尾的，则四个比特'0000'（填充）应被附加在比特流之后，将最后一个八位组补充完整。

12.12 一旦执行了上述步骤，则比特流的内容便是一个快速信息集文档。

附 件 A

快速信息集文档的ASN.1模块和ECN模块

（本附件是本建议书 | 国际标准的组成部分）

A.1 ASN.1模块定义

```
FastInfoset {joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) modules(0)
fast-infoset(0)}
DEFINITIONS AUTOMATIC TAGS ::= BEGIN
finf-doc-opt-decl OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) asn1(1)
      generic-applications(10) fast-infoset(0) encodings(1)
      optional-xml-declaration(0)}
finf-doc-no-decl OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) asn1(1)
      generic-applications(10) fast-infoset(0) encodings(1)
      no-xml-declaration(1)}
Document ::= SEQUENCE {
      additional-data         SEQUENCE (SIZE(1..one-meg)) OF
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } OPTIONAL,
      initial-vocabulary      SEQUENCE {
            external-vocabulary         URI OPTIONAL,
            restricted-alphabets        SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            encoding-algorithms         SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            prefixes                    SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            namespace-names             SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            local-names                 SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-ncnames               SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-uris                  SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            attribute-values            SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            content-character-chunks    SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            other-strings               SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            element-name-surrogates     SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL,
            attribute-name-surrogates   SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL }
                  (CONSTRAINED BY {
                        -- 如果 initial-vocabulary 组件存在，则
                        -- 至少有它的一个组件应当存在 -- }) OPTIONAL,
      notations                     SEQUENCE (SIZE(1..MAX)) OF
                                          Notation OPTIONAL,
      unparsed-entities             SEQUENCE (SIZE(1..MAX)) OF
                                          UnparsedEntity OPTIONAL,
      character-encoding-scheme     NonEmptyOctetString OPTIONAL,
      standalone                    BOOLEAN OPTIONAL,
      version                       NonIdentifyingStringOrIndex OPTIONAL
                                      -- OTHER STRING 类别 --,
      children                      SEQUENCE (SIZE(0..MAX)) OF
            CHOICE {
                  element                     Element,
                  processing-instruction      ProcessingInstruction,
                  comment                     Comment,
                  document-type-declaration DocumentTypeDeclaration }}
```

```
one-meg INTEGER ::= 1048576 - 2 的 20 次方
four-gig INTEGER ::= 4294967296 - 2 的 32 次方
NonEmptyOctetString ::= OCTET STRING (SIZE(1..four-gig))
URI ::= NonEmptyOctetString
Element ::= SEQUENCE {
      namespace-attributes    SEQUENCE (SIZE(1..MAX)) OF
            NamespaceAttribute OPTIONAL,
      qualified-name          QualifiedNameOrIndex
                                    -- ELEMENT NAME 类别 --,
      attributes              SEQUENCE (SIZE(1..MAX)) OF
            Attribute OPTIONAL,
      children                SEQUENCE (SIZE(0..MAX)) OF
            CHOICE {
                  element                       Element,
                  processing-instruction        ProcessingInstruction,
                  unexpanded-entity-reference   UnexpandedEntityReference,
                  character-chunk               CharacterChunk,
                  comment                       Comment }}
Attribute ::= SEQUENCE {
      qualified-name      QualifiedNameOrIndex
                                    -- ATTRIBUTE NAME 类别  --,
      normalized-value    NonIdentifyingStringOrIndex
                                    -- ATTRIBUTE VALUE 类别 -- }
ProcessingInstruction ::= SEQUENCE {
      target       IdentifyingStringOrIndex
                              -- OTHER NCNAME 类别 --,
      content      NonIdentifyingStringOrIndex
                              -- OTHER STRING 类别 -- }
UnexpandedEntityReference ::= SEQUENCE {
      name                 IdentifyingStringOrIndex
                                    -- OTHER NCNAME 类别 --,
      system-identifier    IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI 类别 --,
      public-identifier    IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI 类别 -- }
CharacterChunk ::= SEQUENCE {
      character-codes             NonIdentifyingStringOrIndex
                                          -- CONTENT CHARACTER CHUNK 类别 -- }
Comment ::= SEQUENCE {
      content     NonIdentifyingStringOrIndex -- OTHER STRING 类别 --}
DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier    IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI 类别 --,
      public-identifier    IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI 类别 --,
      children             SEQUENCE (SIZE(0..MAX)) OF
            ProcessingInstruction }
UnparsedEntity ::= SEQUENCE {
      name                 IdentifyingStringOrIndex
                                    -- OTHER NCNAME 类别 --,
      system-identifier    IdentifyingStringOrIndex
                                    -- OTHER URI 类别 --,
      public-identifier    IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI 类别 --,
      notation-name        IdentifyingStringOrIndex
                                    -- OTHER NCNAME 类别 -- }
Notation ::= SEQUENCE {
      name                 IdentifyingStringOrIndex
                                    -- OTHER NCNAME 类别 --,
      system-identifier    IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI 类别 --,
      public-identifier    IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI 类别 -- }
```

```
NamespaceAttribute ::= SEQUENCE {
      prefix             IdentifyingStringOrIndex OPTIONAL
                                 -- PREFIX 类别 --,
      namespace-name     IdentifyingStringOrIndex OPTIONAL
                                 -- NAMESPACE NAME 类别 -- }
IdentifyingStringOrIndex ::= CHOICE {
      literal-character-string   NonEmptyOctetString,
      string-index               INTEGER (1..one-meg) }
NonIdentifyingStringOrIndex ::= CHOICE {
      literal-character-string    SEQUENCE {
            add-to-table               BOOLEAN,
            character-string           EncodedCharacterString },
      string-index                INTEGER (0..one-meg) }
NameSurrogate ::= SEQUENCE {
      prefix-string-index           INTEGER(1..one-meg) OPTIONAL,
      namespace-name-string-index   INTEGER(1..one-meg) OPTIONAL,
      local-name-string-index       INTEGER(1..one-meg) }
      (CONSTRAINED BY  {-- 当且仅当 namespace-name-string-index 存在时,
                        -- prefix-string-index 才应当存在 --})
QualifiedNameOrIndex ::= CHOICE {
      literal-qualified-name  SEQUENCE {
            prefix                IdentifyingStringOrIndex OPTIONAL
                                        -- PREFIX 类别 --,
            namespace-name        IdentifyingStringOrIndex OPTIONAL
                                        -- NAMESPACE NAME 类别 --,
            local-name            IdentifyingStringOrIndex
                                        -- LOCAL NAME 类别 -- },
      name-surrogate-index    INTEGER (1..one-meg) }
EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOICE {
            utf-8                  NULL,
            utf-16                 NULL,
            restricted-alphabet    INTEGER(1..256),
            encoding-algorithm     INTEGER(1..256) },
      octets                NonEmptyOctetString }
END
```

A.2 ECN 模块定义

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

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

```
-- 有用的编码类
#PositiveOrNonNegativeInteger ::= #INTEGER
#NonEmptySequenceOfLength ::= #INT(1..1048576)
#NonEmptyOctetStringLength ::= #INT(1..4294967296)
#TwoAlternativeDiscriminant ::= #INT(0..1)
#ThreeAlternativeDiscriminant ::= #INT(0..2)
#FourAlternativeDiscriminant ::= #INT(0..3)
#FiveAlternativeDiscriminant ::= #INT(0..4)
-- 在对 SEQUENCE OF 的长度进行编码时使用（见 C.21）
#NonEmptySequenceOfLengthAlternatives1 ::= #ALTERNATIVES {
      small         #INT(1..128),
      large         #INT(129..1048576) }
-- 在对 NonEmptyOctetString 的长度进行编码时使用（见 C.22）
#NonEmptyOctetStringLengthAlternatives2 ::= #ALTERNATIVES {
      small         #INT(1..64),
      medium        #INT(65..320),
      large         #INT(321..4294967296) }
-- 在对 NonEmptyOctetString 的长度进行编码时使用（见 C.23）
#NonEmptyOctetStringLengthAlternatives5 ::= #ALTERNATIVES {
      small         #INT(1..8),
      medium        #INT(9..264),
      large         #INT(265..4294967296) }
-- 在对 NonEmptyOctetString 的长度进行编码时使用（见 C.24）
#NonEmptyOctetStringLengthAlternatives7 ::= #ALTERNATIVES {
      small         #INT(1..2),
      medium        #INT(3..258),
      large         #INT(259..4294967296) }
-- 在对一个正整数进行编码时使用（见 C.25）
#PositiveIntegerAlternatives2 ::= #ALTERNATIVES {
      small          #INT(1..64),
      medium         #INT(65..8256),
      large          #INT(8257..1048576) }
-- 在对一个正整数进行编码时使用（见 C.27）
#PositiveIntegerAlternatives3 ::= #ALTERNATIVES {
      small          #INT(1..32),
      medium         #INT(33..2080),
      medium-large   #INT(2081..526368),
      large          #INT(526369..1048576) }
-- 在对一个正整数进行编码时使用（见 C.28）
#PositiveIntegerAlternatives4 ::= #ALTERNATIVES {
      small          #INT(1..16),
      medium         #INT(17..1040),
      medium-large   #INT(1041..263184),
      large          #INT(263185..1048576) }
-- 在对一个非负整数进行编码时使用（见 C.26）
#NonNegativeIntegerAlternatives2 ::= #ALTERNATIVES {
      zero           #INT(0),
      small          #INT(1..64),
      medium         #INT(65..8256),
      large          #INT(8257..1048576) }
-- 在很多情况下，用于在一个编码前插入前导填充字符
#PrecededByPrepadding{<#C>} ::= #CONCATENATION {
      prepadding       #PAD,
      original         #C }
```

```
-- 用于在一个编码前插入两个可替换的判别式
#PrecededByTwoAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      discriminant    #TwoAlternativeDiscriminant,
      original        #C }
-- 用于在一个编码前插入三个可替换的判别式
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      discriminant    #ThreeAlternativeDiscriminant,
      original        #C }
-- 用于在一个编码前插入四个可替换的判别式
#PrecededByFourAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      prepadding      #PAD,
      discriminant    #FourAlternativeDiscriminant,
      original        #C }
-- 用于在一个编码前插入五个可替换的判别式
#PrecededByFiveAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      prepadding      #PAD,
      discriminant    #FiveAlternativeDiscriminant,
      original        #C }
-- 用于在一个 SEQUENCE OF 的编码前插入一个长度字段
#PrecededByNonEmptySequenceOfLength{<#C>} ::= #CONCATENATION {
      length          #NonEmptySequenceOfLength,
      original        #C }
-- 用于在一个 NonEmptyOctetString 的编码前插入一个长度字段
#PrecededByNonEmptyOctetStringLength{<#C>} ::= #CONCATENATION {
      length          #NonEmptyOctetStringLength,
      original        #C }
-- 对 Document 类型的 notations 组件的一个项进行编码（见 C.2.6.1）
eNotationDriver1 #Notation ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNotationPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }
-- 对 Document 类型的 unparsed-entities 组件的一个项进行编码（见 C.2.7.1）
eUnparsedEntityDriver1 #UnparsedEntity ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eUnparsedEntityPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }
-- 对 Element 类型的 namespace-attributes 组件的一个项进行编码（见 C.3.4.2）
eNamespaceAttributeDriver1 #NamespaceAttribute ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNamespaceAttributePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }
-- 对 Element 类型的 attributes 组件的一个项进行编码（见 C.3.6.1）
eAttributeDriver1 #Attribute ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eAttributePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }
-- 对 Document 类型的 attribute-values, content-character-chunks 和 other-strings
-- 组件的一个项进行编码（见 C.2.5.4）
eEncodedCharacterStringDriver1 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eEncodedCharacterStringPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }
```

```
-- 对 Document 类型的 element-name-surrogates 和 attribute-name-surrogates
-- 组件的一个项进行编码（见 C.2.5.5）
eNameSurrogateDriver1 #NameSurrogate ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNameSurrogatePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }
-- 对 Document 类型的 initial-vocabulary 组件进行编码（见 C.2.5）
eInitialVocabularyPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eInitialVocabularyWithPrepadding1 }
-- 在 Document 类型的 notations 组件的一个项的编码前，
-- 插入前导填充（见 C.2.6.1）
eNotationPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNotationWithPrepadding1 }
-- 在 Document 类型的 unparsed-entities 组件的一个项的编码前，
-- 插入前导填充（见 C.2.7.1）
eUnparsedEntityPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eUnparsedEntityWithPrepadding1 }
-- 在 Document 类型的 standalone 组件的一个项的编码前，
-- 插入前导填充（见 C.2.9）
eStandalonePrepaddingAdder1 #BOOL ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eStandaloneWithPrepadding1 }
-- 在 DocumentTypeDeclaration 类型的 children 组件的一个项的编码前，
-- 插入前导填充（见 C.9.6）
eDocTypeDeclChildPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eDocTypeDeclChildWithPrepadding1 }
-- 在 Element 类型的 namespace-attributes 组件的一个项的编码前，
-- 插入前导填充（见 C.3.4.2）
eNamespaceAttributePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNamespaceAttributeWithPrepadding1 }
-- 在 Element 类型的 attributes 组件的一个项的编码前，
-- 插入前导填充（见 C.3.6.1）
eAttributePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eAttributeWithPrepadding1 }
-- 在 QualifiedNameOrIndex 类型的 literal-qualified-name 组件的编码前，
-- 插入前导填充。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.17.3）
eLiteralQualifiedNamePrepaddingAdder2 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eLiteralQualifiedNameWithPrepadding2 }
-- 在 QualifiedNameOrIndex 类型的 literal-qualified-name 组件的编码前，
-- 插入前导填充。
-- 当编码起始于一个八位组的第三个比特时使用（见 C.18.3）
eLiteralQualifiedNamePrepaddingAdder3 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eLiteralQualifiedNameWithPrepadding3 }
```

```
-- 在 Document 类型的 restricted-alphabets, encoding-algorithms, prefixes,
-- namespace-names, local-names, other-ncnames 和 other-uris 组件的
-- 一个项的编码前插入前导填充（见 C.2.5.3）
eNonEmptyOctetStringPrepaddingAdder1 #OCTET-STRING ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByPrepadding
          ENCODED BY eNonEmptyOctetStringWithPrepadding1 }}
-- 在 Document 类型的 attribute-values,content-character-chunks 和 other-strings
-- 组件的一个项的编码前插入前导填充（见 C.2.5.4）
eEncodedCharacterStringPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eEncodedCharacterStringWithPrepadding1 }
-- 在 Document 类型的 element-name-surrogates 和 attribute-name-surrogates 组件
-- 的一个项的编码前插入前导填充（见 C.2.5.5）
eNameSurrogatePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNameSurrogateWithPrepadding1 }
-- 在 Document 类型的 children 组件的一个项的编码前
-- 插入一个判别式（见 C.2.11.2 到 C.2.11.5）
eDocumentChildDiscriminantAdder1or5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY eDocumentChildWithDiscriminant1or5 }
-- 在 Element 类型的 children 组件的一个项的编码前
-- 插入一个判别式（见 C.3.7.2 到 C.3.7.6）
eElementChildDiscriminantAdder1or5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFiveAlternativeDiscriminant
      ENCODED BY eElementChildWithDiscriminant1or5 }
-- 在对 SEQUENCE OF 的长度进行编码前插入一个判别式,
-- 标识长度编码的两种方式之一（见 C.21）
eNonEmptySequenceOfLengthDiscriminantAdder1 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByTwoAlternativeDiscriminant
      ENCODED BY eNonEmptySequenceOfLengthWithDiscriminant1 }
-- 在对 NonEmptyOctetString 的长度进行编码前插入一个判别式,
-- 标识长度编码的三种方式之一。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.22）
eNonEmptyOctetStringLengthDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant2 }
-- 在对 NonEmptyOctetString 的长度进行编码前插入一个判别式,
-- 标识长度编码的三种方式之一。
-- 当编码起始于一个八位组的第五个比特时使用（见 C.23）
eNonEmptyOctetStringLengthDiscriminantAdder5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant5 }
-- 在对 NonEmptyOctetString 的长度进行编码前插入一个判别式,
-- 标识长度编码的三种方式之一。
-- 当编码起始于一个八位组的第七个比特时使用（见 C.24）
eNonEmptyOctetStringLengthDiscriminantAdder7 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant7 }
```

```
-- 在对一个正整数进行编码前插入一个判别式，
-- 标识对其进行编码的三种方式之一。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.25）
ePositiveIntegerDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant2 }
-- 在对一个正整数进行编码前插入一个判别式，
-- 标识对其进行编码的四种方式之一。
-- 当编码起始于一个八位组的第三个比特时使用（见 C.27）
ePositiveIntegerDiscriminantAdder3 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant3 }
-- 在对一个正整数进行编码前插入一个判别式，
-- 标识对其进行编码的四种方式之一。
-- 当编码起始于一个八位组的第四个比特时使用（见 C.28）
ePositiveIntegerDiscriminantAdder4 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant4 }
-- 在对一个非负整数进行编码前插入一个判别式，
-- 标识对其进行编码的三种方式之一（见 C.26）
eNonNegativeIntegerDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY eNonNegativeIntegerWithDiscriminant2 }
-- 对已经加入到 Document 类型的 initial-vocabulary 组件前的前导填充进行设置，
-- 并且对该组件进行编码（见 C.2.5）
eInitialVocabularyWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 3
              PAD-PATTERN bits:'000'B },
          original {
              ENCODE STRUCTURE {
                  restricted-alphabets {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH
      eRepetitionWithLengthNonEmptyOctetString1 }
                          WITH FastInfosetEncodingSet }
                              OPTIONAL-ENCODING USE-SET,
                  encoding-algorithms {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH
      eRepetitionWithLengthNonEmptyOctetString1 }
                          WITH FastInfosetEncodingSet }
                              OPTIONAL-ENCODING USE-SET,
                  prefixes {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH
      eRepetitionWithLengthNonEmptyOctetString1 }
                          WITH FastInfosetEncodingSet }
                              OPTIONAL-ENCODING USE-SET,
                  namespace-names {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH
      eRepetitionWithLengthNonEmptyOctetString1 }
                          WITH FastInfosetEncodingSet }
                              OPTIONAL-ENCODING USE-SET,
                  local-names {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH
      eRepetitionWithLengthNonEmptyOctetString1 }
                          WITH FastInfosetEncodingSet }
                              OPTIONAL-ENCODING USE-SET,
                  other-ncnames {
                      ENCODE STRUCTURE {
```

```
                        STRUCTURED WITH
      eRepetitionWithLengthNonEmptyOctetString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                other-uris {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
      eRepetitionWithLengthNonEmptyOctetString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                attribute-values {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                content-character-chunks {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                other-strings {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                element-name-surrogates {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH eRepetitionWithLengthNameSurrogate1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                attribute-name-surrogates {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH eRepetitionWithLengthNameSurrogate1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET }
            WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }
-- 对已经加入到 Document 类型的 notations 组件每一项前的前导填充进行设置，
-- 并且对该项进行编码（见 C.2.6.1）
eNotationWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 6
              PAD-PATTERN bits:'110000'B },
          original eNotation7 }
      WITH FastInfosetEncodingSet }
-- 对已经加入到 Document 类型的 unparsed-entities 组件每一项前的前导填充进行设置，
-- 并且对该项进行编码（见 C.2.7.1）
eUnparsedEntityWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 7
              PAD-PATTERN bits:'1101000'B },
          original eUnparsedEntity8 }
      WITH FastInfosetEncodingSet }
-- 对已经加入到 Document 类型的 standalone 组件前的前导填充进行设置，
-- 并且对该组件进行编码（见 C.2.9）
eStandaloneWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 7
              PAD-PATTERN bits:'0000000'B },
          original USE-SET }
```

```
        WITH FastInfosetEncodingSet }
    -- 对已经加入到 Element 类型的 namespace-attributes 组件每一项前的前导填充进行设置，
    -- 并且对该项进行编码（见 C.3.4.2）
    eNamespaceAttributeWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
        ENCODE STRUCTURE {
            prepadding {
                ENCODING-SPACE SIZE 6
                PAD-PATTERN bits:'110011'B
                EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5}
                AS bits:'110011'B },
            original eNamespaceAttribute7
            STRUCTURED WITH {
                ENCODING-SPACE
                EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5} AS bits:'110011'B }}
        WITH FastInfosetEncodingSet }
    -- 对已经加入到 Element 类型的 attributes 组件每一项前的前导填充进行设置，
    -- 并且对该项进行编码（见 C.3.6.1）
    eAttributeWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
        ENCODE STRUCTURE {
            prepadding {
                ENCODING-SPACE SIZE 1
                PAD-PATTERN bits:'0'B },
            original eAttribute2 }
        WITH FastInfosetEncodingSet }
    -- 对已经加入到 DocumentTypeDeclaration 类型的 children 组件每一项前的前导填充进行设置，
    -- 并且对该项进行编码（见 C.9.6）
    eDocTypeDeclChildWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
        ENCODE STRUCTURE {
            prepadding {
                ENCODING-SPACE SIZE 8
                PAD-PATTERN bits:'11100001'B },
            original eProcessingInstruction1 }
        WITH FastInfosetEncodingSet }
    -- 对已经加入到 Document 类型的 restricted-alphabets, encoding-algorithms, prefixes,
    -- namespace-names, local-names, other-ncnames 和 other-uris 组件
    -- 每一项前的前导填充进行设置，并且对该项进行编码（见 C.2.5.3）
    eNonEmptyOctetStringWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
        ENCODE STRUCTURE {
            prepadding {
                ENCODING-SPACE SIZE 1
                PAD-PATTERN bits:'0'B },
            original USE-SET }
        WITH FastInfosetEncodingSet }
    -- 对已经加入到 Document 类型的 attribute-values, content-character-chunks 和
    -- other-strings 组件每一项前的前导填充进行设置，
    -- 并且对该项进行编码（见 C.2.5.4）
    eEncodedCharacterStringWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
        ENCODE STRUCTURE {
            prepadding {
                ENCODING-SPACE SIZE 2
                PAD-PATTERN bits:'00'B },
            original USE-SET }
        WITH FastInfosetEncodingSet }

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

```
-- 对已经加入到 QualifiedNameOrIndex 类型的 literal-qualified-name 组件
-- 前的前导填充进行设置，并且对该组件进行编码。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.17.3）
eLiteralQualifiedNameWithPrepadding2{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 5
              PAD-PATTERN bits:'11110'B },
          original {
              ENCODE STRUCTURE {
                  prefix eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  namespace-name eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  local-name eIdentifyingStringOrIndex1 }
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
              ENCODING-SPACE
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B }}
      WITH FastInfosetEncodingSet }
-- 对已经加入到 QualifiedNameOrIndex 类型的 literal-qualified-name 组件
-- 前的前导填充进行设置，并且对该组件进行编码。
-- 当编码起始于一个八位组的第三个比特时使用（见 C.18.3）
eLiteralQualifiedNameWithPrepadding3{<#C>}
#PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 4
              PAD-PATTERN bits:'1111'B
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B },
          original {
              ENCODE STRUCTURE {
                  prefix eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  namespace-name eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  local-name eIdentifyingStringOrIndex1 }
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
              ENCODING-SPACE
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B }}
      WITH FastInfosetEncodingSet }
-- 对已经加入到一个 SEQUENCE OF 前的长度字段进行编码，
-- 并且对 SEQUENCE OF NonEmptyOctetString 进行编码（见 C.21）
eNonEmptySequenceOfWithLengthNonEmptyOctetString1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eNonEmptyOctetStringPrepaddingAdder1
                  STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }
-- 对已经加入到一个 SEQUENCE OF 前的长度字段进行编码，
-- 并且对 SEQUENCE OF EncodedCharacterString 进行编码（见 C.21）
eNonEmptySequenceOfWithLengthEncodedCharacterString1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eEncodedCharacterStringDriver1
                  STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }
```

```
-- 对已经加入到一个 SEQUENCE OF 前的长度字段进行编码，
-- 并且对 SEQUENCE OF NameSurrogate 进行编码（见 C.21）
eNonEmptySequenceOfWithLengthNameSurrogate1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eNameSurrogateDriver1
                 STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }
-- 对已经加入到一个 SEQUENCE OF 前的长度字段进行编码，
-- 并且对 SEQUENCE OF additional-datum 进行编码（见 C.21）
eNonEmptySequenceOfWithLengthAdditionalDatum1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  additional-datum {
                       ENCODE STRUCTURE {
                            id eNonEmptyOctetStringPrepaddingAdder1,
                            data eNonEmptyOctetStringPrepaddingAdder1 }
                       WITH FastInfosetEncodingSet}
                 STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }
-- 对已经加入到一个 NonEmptyOctetString 前的长度字段进行编码，
-- 并且对 NonEmptyOctetString 进行编码。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.22）
eNonEmptyOctetStringWithLength2{<#C>}
#PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength2,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }
-- 对已经加入到一个 NonEmptyOctetString 前的长度字段进行编码，
-- 并且对 NonEmptyOctetString 进行编码。
-- 当编码起始于一个八位组的第五个比特时使用（见 C.23）
eNonEmptyOctetStringWithLength5{<#C>}
      #PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength5,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }
-- 对已经加入到一个 NonEmptyOctetString 前的长度字段进行编码，
-- 并且对 NonEmptyOctetString 进行编码。
-- 当编码起始于一个八位组的第七个比特时使用（见 C.24）
eNonEmptyOctetStringWithLength7{<#C>}
#PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength7,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }
-- 对已经加入到 Document 类型的 children 组件每一项前的判别式进行编码，
-- 并且对该项进行编码（见 C.2.11.2 到 C.2.11.5）
eDocumentChildWithDiscriminant1or5{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ALIGNED TO NEXT octet
              ENCODING-SPACE SIZE 0 },
          discriminant {
```

```
            USE #BIT-STRING
             MAPPING TO BITS {
                0 TO '0'B,
                1 TO '11100001'B,
                2 TO '11100010'B,
                3 TO '110001'B }
            WITH FastInfosetEncodingSet },
        original {
            ENCODE STRUCTURE {
                element eElement2,
                processing-instruction eProcessingInstruction1,
                comment eComment1,
                document-type-declaration eDocumentTypeDeclaration7
                STRUCTURED WITH {
                    ALTERNATIVE DETERMINED BY field-to-be-set
                    USING discriminant }}
            WITH FastInfosetEncodingSet }}
    WITH FastInfosetEncodingSet }
-- 对已经加入到 Element 类型的 children 组件每一项前的判别式进行编码，
-- 并且对该项进行编码（见 C.3.7.2 到 C.3.7.6）
eElementChildWithDiscriminant1or5{<#C>}
#PrecededByFiveAlternativeDiscriminant{<#C>} ::= {
    ENCODE STRUCTURE {
        prepadding {
            ALIGNED TO NEXT octet
            ENCODING-SPACE SIZE 0 },
        discriminant {
            USE #BIT-STRING
            MAPPING TO BITS {
                0 TO '0'B,
                1 TO '11100001'B,
                2 TO '110010'B,
                3 TO '10'B,
                4 TO '11100010'B }
            WITH FastInfosetEncodingSet },
        original {
            ENCODE STRUCTURE {
                element eElement2,
                processing-instruction eProcessingInstruction1,
                unexpanded-entity-reference eUnexpandedEntityReference7,
                character-chunk eCharacterChunk3,
                comment eComment1
                STRUCTURED WITH {
                    ALTERNATIVE DETERMINED BY field-to-be-set
                    USING discriminant }}
            WITH FastInfosetEncodingSet }}
    WITH FastInfosetEncodingSet }
-- 对已经加入到 SEQUENCE OF 的长度前的判别式进行编码，
-- （该判别式标识了对长度进行编码的两种方式之一）
-- 并且对此长度进行编码（见 C.21）
eNonEmptySequenceOfLengthWithDiscriminant1{<#C>}
#PrecededByTwoAlternativeDiscriminant{<#C>} ::= {
    ENCODE STRUCTURE {
        discriminant {
            USE #BIT-STRING
            MAPPING TO BITS {
                0 TO '0'B,
                1 TO '1000'B }
            WITH FastInfosetEncodingSet },
        original {
            ENCODE STRUCTURE {
                STRUCTURED WITH {
                    ALTERNATIVE DETERMINED BY field-to-be-set
                    USING discriminant }}
            WITH FastInfosetEncodingSet }}
    WITH FastInfosetEncodingSet }
```

```
-- 对已经加入到 NonEmptyOctetString 的长度前的判别式进行编码，
-- （该判别式标识了对长度进行编码的三种方式之一），并且对此长度进行编码。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.22）
eNonEmptyOctetStringLengthWithDiscriminant2{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '1000000'B,
                  2 TO '1100000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }
-- 对已经加入到 NonEmptyOctetString 的长度前的判别式进行编码，
-- （该判别式标识了对长度进行编码的三种方式之一），并且对此长度进行编码。
-- 当编码起始于一个八位组的第五个比特时使用（见 C.23）
eNonEmptyOctetStringLengthWithDiscriminant5{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '1000'B,
                  2 TO '1100'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }
-- 对已经加入到 NonEmptyOctetString 的长度前的判别式进行编码，
-- （该判别式标识了对长度进行编码的三种方式之一），并且对此长度进行编码。
-- 当编码起始于一个八位组的第七个比特时使用（见 C.24）
eNonEmptyOctetStringLengthWithDiscriminant7{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '10'B,
                  2 TO '11'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }
```

```
-- 对已经加入到一个正整数前的判别式进行编码（该判别式标识了对整数进行编码的三种方式之一），
-- 并且对此整数进行编码。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.25）
ePositiveIntegerWithDiscriminant2{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '10'B,
                  2 TO '110'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
                  ENCODING-SPACE SIZE self-delimiting-values
                  EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 }
                  AS range:{low 0, high 12}}}  -- Less than '1110'B
      WITH FastInfosetEncodingSet }
-- 对已经加入到一个正整数前的判别式进行编码（该判别式标识了对整数进行编码的三种方式之一），
-- 并且对此整数进行编码。
-- 当编码起始于一个八位组的第三个比特时使用（见 C.27）
ePositiveIntegerWithDiscriminant3{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '100'B,
                  2 TO '101'B,
                  3 TO '110000000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
                  ENCODING-SPACE SIZE self-delimiting-values
                  EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 }
                  AS range:{low 0, high 14}}}  -- Less than '1111'B
      WITH FastInfosetEncodingSet }
-- 对已经加入到一个正整数前的判别式进行编码（该判别式标识了对整数进行编码的四种方式之一），
-- 并且对此整数进行编码。
-- 当编码起始于一个八位组的第四个比特时使用（见 C.28）
ePositiveIntegerWithDiscriminant4{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '100'B,
                  2 TO '101'B,
                  3 TO '110000000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
```

```
                    STRUCTURED WITH {
                        ALTERNATIVE DETERMINED BY field-to-be-set
                        USING discriminant }}
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }
-- 对已经加入到一个非负整数前的判别式进行编码（该判别式标识了对整数进行编码的三种方式之一），
-- 并且对此整数进行编码（见 C.26）
eNonNegativeIntegerWithDiscriminant2{<#C>} #PrecededByFourAlternativeDiscriminant{<#C>}
::= {
        ENCODE STRUCTURE {
            discriminant {
                USE #BIT-STRING
                MAPPING TO BITS {
                    0 TO '1111111'B,
                    1 TO '0'B,
                    2 TO '10'B,
                    3 TO '110'B }
                WITH FastInfosetEncodingSet },
            original {
                ENCODE STRUCTURE {
                    STRUCTURED WITH {
                        ALTERNATIVE DETERMINED BY field-to-be-set
                        USING discriminant }}
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }
-- 对 Document 类型进行编码（见 C.2）
eDocument2 #Document ::= {
        ENCODE STRUCTURE {
            additional-data {
                ENCODE STRUCTURE {
                    STRUCTURED WITH eRepetitionWithLengthAdditionalDatum1 }
                WITH FastInfosetEncodingSet }
                    OPTIONAL-ENCODING USE-SET,
            initial-vocabulary {
                ENCODE STRUCTURE {
                    STRUCTURED WITH eInitialVocabularyPrepaddingAdder1 }
                WITH FastInfosetEncodingSet }
                    OPTIONAL-ENCODING USE-SET,
            notations {
                ENCODE STRUCTURE {
                    eNotationDriver1
                    STRUCTURED WITH eRepetitionWithTerminator8bit1 }
                WITH FastInfosetEncodingSet }
                    OPTIONAL-ENCODING USE-SET,
            unparsed-entities {
                ENCODE STRUCTURE {
                    eUnparsedEntityDriver1
                    STRUCTURED WITH eRepetitionWithTerminator8bit1 }
                WITH FastInfosetEncodingSet }
                    OPTIONAL-ENCODING USE-SET,
            character-encoding-scheme eNonEmptyOctetStringPrepaddingAdder1
                OPTIONAL-ENCODING USE-SET,
            standalone eStandalonePrepaddingAdder1
                OPTIONAL-ENCODING USE-SET,
            version eNonIdentifyingStringOrIndex1
                OPTIONAL-ENCODING USE-SET,
            children {
                ENCODE STRUCTURE {
                    {
                        ENCODE STRUCTURE {
                            STRUCTURED WITH eDocumentChildDiscriminantAdder1or5 }
                        WITH FastInfosetEncodingSet }
                    STRUCTURED WITH eRepetitionWithTerminator4bit1 }
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }
```

```
-- 对 Element 类型进行编码（见 C.3）
eElement2 #Element ::= {
      ENCODE STRUCTURE {
          namespace-attributes {
              ENCODE STRUCTURE {
                  eNamespaceAttributeDriver1
                  STRUCTURED WITH eRepetitionWithTerminator10bit1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING eNamespaceAttributesOptionality3,
          qualified-name eQualifiedNameOrIndex3,
          attributes {
              ENCODE STRUCTURE {
                  eAttributeDriver1
                  STRUCTURED WITH eRepetitionWithTerminator4bit1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING USE-SET,
          children {
              ENCODE STRUCTURE {
                  {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eElementChildDiscriminantAdder1or5 }
                      WITH FastInfosetEncodingSet }
                  STRUCTURED WITH eRepetitionWithTerminator4bit1 }
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }
-- 对 Attribute 类型进行编码（见 C.4）
eAttribute2 #Attribute ::= {
      ENCODE STRUCTURE {
          qualified-name eQualifiedNameOrIndex2,
          normalized-value eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }
-- 对 ProcessingInstruction 类型进行编码（见 C.5）
eProcessingInstruction1 #ProcessingInstruction ::= {
      ENCODE STRUCTURE {
          target eIdentifyingStringOrIndex1,
          content eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }
-- 对 UnexpandedEntityReference 类型进行编码（见 C.6）
eUnexpandedEntityReference7 #UnexpandedEntityReference ::= {
      ENCODE STRUCTURE {
          name eIdentifyingStringOrIndex1,
          system-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET,
          public-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET }
      WITH FastInfosetEncodingSet }
-- 对 CharacterChunk 类型进行编码（见 C.7）
eCharacterChunk3 #CharacterChunk ::= {
      ENCODE STRUCTURE {
          character-codes eNonIdentifyingStringOrIndex3 }
      WITH FastInfosetEncodingSet }
-- 对 Comment 类型进行编码（见 C.8）
eComment1 #Comment ::= {
      ENCODE STRUCTURE {
          content eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }
-- 对 DocumentTypeDeclaration 类型进行编码（见 C.9）
eDocumentTypeDeclaration7 #DocumentTypeDeclaration ::= {
      ENCODE STRUCTURE {
          system-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET,
          public-identifier eIdentifyingStringOrIndex1
```

```
                OPTIONAL-ENCODING USE-SET,
            children {
                ENCODE STRUCTURE {
                    {
                        ENCODE STRUCTURE {
                            STRUCTURED WITH eDocTypeDeclChildPrepaddingAdder1 }
                        WITH FastInfosetEncodingSet }
                    STRUCTURED WITH eRepetitionWithTerminator4bit1 }
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }
-- 对 UnparsedEntity 类型进行编码（见 C.10）
eUnparsedEntity8 #UnparsedEntity ::= {
        ENCODE STRUCTURE {
            name eIdentifyingStringOrIndex1,
            system-identifier eIdentifyingStringOrIndex1,
            public-identifier eIdentifyingStringOrIndex1
                OPTIONAL-ENCODING USE-SET,
            notation-name eIdentifyingStringOrIndex1 }
        WITH FastInfosetEncodingSet }
-- 对 Notation 类型进行编码（见 C.11）
eNotation7 #Notation ::= {
        ENCODE STRUCTURE {
            name eIdentifyingStringOrIndex1,
            system-identifier eIdentifyingStringOrIndex1
                OPTIONAL-ENCODING USE-SET,
            public-identifier eIdentifyingStringOrIndex1
                OPTIONAL-ENCODING USE-SET }
        WITH FastInfosetEncodingSet }
-- 对 NamespaceAttribute 类型进行编码（见 C.12）
eNamespaceAttribute7 #NamespaceAttribute ::= {
        ENCODE STRUCTURE {
            prefix eIdentifyingStringOrIndex1
                OPTIONAL-ENCODING USE-SET,
            namespace-name eIdentifyingStringOrIndex1
                OPTIONAL-ENCODING USE-SET }
        WITH FastInfosetEncodingSet }
-- 对 IdentifyingStringOrIndex 类型进行编码（见 C.13）
eIdentifyingStringOrIndex1 #IdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string eNonEmptyOctetString2,
            string-index ePositiveInteger2 }
        WITH FastInfosetEncodingSet }
-- 对 NonIdentifyingStringOrIndex 类型进行编码。
-- 当编码起始于一个八位组的第一个比特时使用（见 C.14）
eNonIdentifyingStringOrIndex1 #NonIdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string {
                ENCODE STRUCTURE {
                    add-to-table USE-SET,
                    character-string eEncodedCharacterString3 }
                WITH FastInfosetEncodingSet },
            string-index eNonNegativeInteger2 }
        WITH FastInfosetEncodingSet }
-- 对 NonIdentifyingStringOrIndex 类型进行编码。
-- 当编码起始于一个八位组的第三个比特时使用（见 C.15）
eNonIdentifyingStringOrIndex3 #NonIdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string {
                ENCODE STRUCTURE {
                    add-to-table USE-SET,
                    character-string eEncodedCharacterString5 }
                WITH FastInfosetEncodingSet },
            string-index ePositiveInteger4 }
```

```
      WITH FastInfosetEncodingSet }
-- 对 NameSurrogate 类型进行编码（见 C.16）
eNameSurrogate7 #NameSurrogate ::= {
      ENCODE STRUCTURE {
          prefix-string-index ePositiveInteger2
                  OPTIONAL-ENCODING USE-SET,
          namespace-name-string-index ePositiveInteger2
                  OPTIONAL-ENCODING USE-SET,
          local-name-string-index ePositiveInteger2 }
      WITH FastInfosetEncodingSet }
-- 对 QualifiedNameOrIndex 类型进行编码。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.17）
eQualifiedNameOrIndex2 #QualifiedNameOrIndex ::= {
      ENCODE STRUCTURE {
          literal-qualified-name {
              ENCODE STRUCTURE {
                  STRUCTURED WITH eLiteralQualifiedNamePrepaddingAdder2 }
              WITH FastInfosetEncodingSet },
          name-surrogate-index ePositiveInteger2
          STRUCTURED WITH eQualifiedNameAlternatives3 }
      WITH FastInfosetEncodingSet }
-- 对 QualifiedNameOrIndex 类型进行编码。
-- 当编码起始于一个八位组的第三个比特时使用（见 C.18）
eQualifiedNameOrIndex3 #QualifiedNameOrIndex ::= {
      ENCODE STRUCTURE {
          literal-qualified-name {
              ENCODE STRUCTURE {
                  STRUCTURED WITH eLiteralQualifiedNamePrepaddingAdder3 }
              WITH FastInfosetEncodingSet },
          name-surrogate-index ePositiveInteger3
          STRUCTURED WITH eQualifiedNameAlternatives3 }
      WITH FastInfosetEncodingSet }
-- 对 EncodedCharacterString 类型进行编码。
-- 当编码起始于一个八位组的第三个比特时使用（见 C.19）
eEncodedCharacterString3 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          encoding-format USE-SET,
          octets eNonEmptyOctetString5 }
      WITH FastInfosetEncodingSet }
-- 对 EncodedCharacterString 类型进行编码。
-- 当编码起始于一个八位组的第五个比特时使用（见 C.20）
eEncodedCharacterString5 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          encoding-format USE-SET,
          octets eNonEmptyOctetString7 }
      WITH FastInfosetEncodingSet }
-- 通过在一个重复类型（SEQUENCE OF NonEmptyOctetString）前插入一个长度字段
-- 对该类型进行编码（见 C.2.5.3 到 C.2.5.5）
eRepetitionWithLengthNonEmptyOctetString1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthNonEmptyOctetString1 }}
-- 通过在一个重复类型（SEQUENCE OF EncodedCharacterString）前插入一个长度字段
-- 对该类型进行编码（见 C.2.5.3 到 C.2.5.5）
eRepetitionWithLengthEncodedCharacterString1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthEncodedCharacterString1 }}
```

```
-- 通过在一个重复类型（SEQUENCE OF NameSurrogate）前插入一个长度字段
-- 对该类型进行编码（见 C.2.5.3 到 C.2.5.5）
eRepetitionWithLengthNameSurrogate1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthNameSurrogate1 }}
-- 通过在一个重复类型（SEQUENCE OF additional-datum）前插入一个长度字段
-- 对该类型进行编码（见 C.2.5.3 到 C.2.5.5）
eRepetitionWithLengthAdditionalDatum1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthAdditionalDatum1 }}
-- 对 NonEmptyOctetString 类型进行编码。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.22）
eNonEmptyOctetString2 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength2 }}
-- 对 NonEmptyOctetString 类型进行编码。
-- 当编码起始于一个八位组的第五个比特时使用（见 C.23）
eNonEmptyOctetString5 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength5 }}
-- 对 NonEmptyOctetString 类型进行编码。
-- 当编码起始于一个八位组的第七个比特时使用（见 C.24）
eNonEmptyOctetString7 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength7 }}
-- 对在 SEQUENCE OF 编码前已经插入的长度字段进行编码
-- （见 C.21）
eNonEmptySequenceOfLength1 #NonEmptySequenceOfLength ::= {
      USE #NonEmptySequenceOfLengthAlternatives1
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptySequenceOfLengthDiscriminantAdder1 }
          WITH FastInfosetEncodingSet }}
-- 对在 NonEmptyOctetString 编码前已经插入的长度字段进行编码，
-- 当编码起始于一个八位组的第二个比特时使用（见 C.22）
eNonEmptyOctetStringLength2 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}
-- 对在 NonEmptyOctetString 编码前已经插入的长度字段进行编码，
-- 当编码起始于一个八位组的第五个比特时使用（见 C.23）
eNonEmptyOctetStringLength5 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives5
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder5 }
          WITH FastInfosetEncodingSet }}
```

```
-- 对在 NonEmptyOctetString 编码前已经插入的长度字段进行编码，
-- 当编码起始于一个八位组的第七个比特时使用（见 C.24）
eNonEmptyOctetStringLength7 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives7
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder7 }
          WITH FastInfosetEncodingSet }}
-- 对一个正整数进行编码。
-- 当编码起始于一个八位组的第二个比特时使用（见 C.25）
ePositiveInteger2 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}
-- 对一个正整数进行编码。
-- 当编码起始于一个八位组的第三个比特时使用（见 C.27）
ePositiveInteger3 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives3
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder3 }
          WITH FastInfosetEncodingSet }}
-- 对一个正整数进行编码。
-- 当编码起始于一个八位组的第四个比特时使用（见 C.28）
ePositiveInteger4 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives4
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder4 }
          WITH FastInfosetEncodingSet }}
-- 对一个非负整数进行编码（见 C.26）
eNonNegativeInteger2 #PositiveOrNonNegativeInteger ::= {
      USE #NonNegativeIntegerAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonNegativeIntegerDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}
-- 规定如何判断 Element 类型中 namespace-attributes 组件的存在
-- （见 C.3.4.2）
eNamespaceAttributesOptionality3 #OPTIONAL ::= {
      PRESENCE DETERMINED BY handle
      HANDLE "nsa" }
-- 规定如何判断 QualifiedNameOrIndex 类型中的可选项
-- （见 C.17.3 和 C.18.3）
eQualifiedNameAlternatives3 #ALTERNATIVES ::= {
      ALTERNATIVE DETERMINED BY handle
      HANDLE "qn"
      EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5}
      AS range:{low 0, high 50}}   -- Less than '110011'B
-- 规定如何使用一个 4 比特的终止符'1111'来判断一个重复类型的结束
-- （见 C.2.12, C.3.6.2, C.3.8 和 C.9.7）
```

```
eRepetitionWithTerminator4bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'1111'B }}
-- 规定如何使用一个 8 比特的终止符'11110000'来判断一个重复类型的结束
-- （见 C.2.6.2 和 C.2.7.2）
eRepetitionWithTerminator8bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'11110000'B }}
-- 规定如何使用一个 10 比特的终止符'1111000000'来判断一个重复类型的结束
-- （见 C.3.4.3）
eRepetitionWithTerminator10bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'1111000000'B
          EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5} AS bits:'110011'B }}
-- 对一个 SEQUENCE OF 类型中的项进行编码，跟在已经加入的长度字段的编码后
-- （见 C.2.5.3 到 C.2.5.5）
eRepetitionItems1{<REFERENCE:len>} #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant MULTIPLE OF bit
          DETERMINED BY field-to-be-set USING len }}
-- 对一个 NonEmptyOctetString 类型中的八位组进行编码，跟在已经加入的长度字段的编码后
-- （见 C.22，C.23 和 C.24）
eOctetStringOctets1{<REFERENCE:len>} #OCTETS ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant MULTIPLE OF bit
          DETERMINED BY field-to-be-set USING len }}
empty-padding #PAD ::= {
       ENCODING-SPACE SIZE 0
}
FastInfosetEncodingSet #ENCODINGS::= { eDocument2 | empty-padding }
      COMPLETED BY PER-BASIC-UNALIGNED
END
FastInfosetELM
      {joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) modules(0)
      fast-infoset-elm(2)}
      LINK-DEFINITIONS ::= BEGIN
      IMPORTS FastInfosetEncodingSet, Document FROM FastInfosetEDM;
      ENCODE #Document WITH FastInfosetEncodingSet
END
```

附 件 B

快速信息集文档的MIME媒体类型

（本附件是本建议书 | 国际标准的组成部分）

本附件定义了描述快速信息集文档的“应用/快速信息集”媒体类型。

MIME 媒体类型在下面规定，使用了 IETF 的 MIME 注册模板，并且已经根据 IETF 的过程进行了注册。

MIME 媒体类型名字：

application

MIME 子类型名字：

fastinfoset

要求参数：

无。

可选参数：

无。

编码考虑：

将 XML 信息集编码为一个快速信息集文档的结果是产生一个二进制数据。这种 MIME 媒体类型可能会对不能处理二进制数据的传送提出更进一步的编码要求。

安全考虑：

由于编码为一个快速信息集文档的 XML 信息集能够携带应用所定义的数据，其语义是独立于任何 MIME 外套的语义的（或者独立于使用 MIME 外套的任何上下文），人们不应期望仅仅通过 MIME 外套的语义就能够理解快速信息集文档的语义。因此，无论何时使用"应用/快速信息集"媒体类型，都强烈建议在使用快速信息集文档的上下文内，其所隐含的安全应当被充分地理解。

协作考虑：

目前尚无已知的协作性问题。

已出版的规范：

ITU-T X.891 建议书 | ISO/IEC 24824-1

使用此媒体类型的应用：

目前，尚未知使用此媒体类型的已知应用。

其他信息：

Magic number(s):

一个快速信息集文档可以起始于一个可选的 XML 声明，应当是下面以 UTF-8 编码的字符串之一：

```
<?xml encoding='finf'?>
<?xml encoding='finf' standalone='yes'?>
<?xml encoding='finf' standalone='no'?>
<?xml version='1.0' encoding='finf'?>
<?xml version='1.0' encoding='finf' standalone='yes'?>
<?xml version='1.0' encoding='finf' standalone='no'?>
<?xml version='1.1' encoding='finf'?>
<?xml version='1.1' encoding='finf' standalone='yes'?>
<?xml version='1.1' encoding='finf' standalone='no'?>
```

以 UTF-8 编码的 XML 声明中的前五个八位组为十六进制的 3C 3F 78 6D 6C。标识一个快速信息集文档的 4 个八位组，对应于以 UTF-8 编码的子串"finf"，为十六进制的 66 69 6E 66。

如果可选的 XML 声明不存在，则一个快速信息集文档应当起始于一个八位组序列，其十六进制表示为 E0 00 00 01。

文件扩展：

*.finf

联络人和联络邮件，可获取后续信息：

ITU-T ASN.1 报告人（通过 tsbmail@itu.int来联系）

ISO/IEC JTC1/SC6 ASN.1 报告人（通过 ittf@iso.org来联系）

预期使用：

公用

作者/修改控制者：

Joint ITU-T | ISO/IEC balloting procedures in accordance with ITU-T Rec. A.23 *Collaboration with the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) on information technology*, Annex A and ISO/IEC JTC1 Directives, Annex K.

附 件 C

快速信息集文档编码的描述

（本附件不是本建议书 | 国际标准的组成部分）

C.1 快速信息集文档

C.1.1 本附件非正式地（但是精确并完整地）描述了第 12 节和附件 A 所规定的编码。为了便于实现者的实现，所有的以规范文本描述的 ASN.1 类型定义都拷贝在本附件中，而不仅仅是简单的引用。.

C.1.2 对编码的描述是通过一个编码者所执行的动作来完成的，结果是产生一些将被附加在一个比特流后面的比特。解码者所要执行的动作在本附件中没有被显式地描述，但是可以通过在本附件中描述的编码者的动作推导出来。

C.1.3 一个快速信息集文档可能或者起始于一个 XML 声明（见 12.3 节），并在后面跟随：

a） 16个比特 '1110000000000000'（标识）；后面跟随

b） 16个比特 '0000000000000001'（版本号）；后面跟随

c） 比特'0' （填充）。

或者起始于上述相同的 33 个比特，但之前没有 XML 声明。33 个比特后紧接着是对 **Document** 类型的一个抽象值的编码，如同 C.2 中的描述。此编码结束于一个八位组的第八个比特或者第四个比特，依赖于快速信息集文档的内容。在后一种情况下，4 个比特'0000'（填充）将被附加在比特流之后。

C.2 Document 类型的编码

C.2.1 **Document** 类型在 7.2 节定义，如下所述：

```
Document ::= SEQUENCE {
      additional-data          SEQUENCE (SIZE(1..one-meg)) OF
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } OPTIONAL,
      initial-vocabulary       SEQUENCE {
            external-vocabulary         URI OPTIONAL,
            restricted-alphabets        SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            encoding-algorithms         SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            prefixes                    SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            namespace-names             SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            local-names                 SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-ncnames               SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-uris                  SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            attribute-values            SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            content-character-chunks    SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            other-strings               SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            element-name-surrogates     SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL,
            attribute-name-surrogates   SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL }
                  (CONSTRAINED BY {
                        -- 如果 initial-vocabulary 组件存在，
                        -- 则至少应当有它的一个子组件存在 -- }) OPTIONAL,
      notations                     SEQUENCE (SIZE(1..MAX)) OF
                                          Notation OPTIONAL,
      unparsed-entities             SEQUENCE (SIZE(1..MAX)) OF
                                          UnparsedEntity OPTIONAL,
```

```
character-encoding-scheme  NonEmptyOctetString OPTIONAL,
standalone                 BOOLEAN OPTIONAL,
version                    NonIdentifyingStringOrIndex OPTIONAL
                             -- OTHER STRING 类别 --,
children                   SEQUENCE (SIZE(0..MAX)) OF
      CHOICE {
            element                    Element,
            processing-instruction     ProcessingInstruction,
            comment                    Comment,
            document-type-declaration DocumentTypeDeclaration }}
```

C.2.2 **Document** 类型的一个值通过（按顺序）执行下列动作来进行编码。

注 — 此类型的一个编码总是起始于一个八位组的第二个比特，并且终止于另一个八位组的第四个或第八个比特（终止于C.2.12所描述的终止符'1111'的最后一个比特）。

C.2.3 对于七个可选组件 **additional-data**，**initial-vocabulary**，**notations**，**unparsed-entities**，**character-encoding-scheme**，**standalone** 和 **version**（按此顺序）中的每一个，如果该组件存在，则比特'1'（表示存在）将被附加在比特流之后，否则比特'0'（表示缺失）将被附加在比特流之后。

C.2.4 如果可选组件 **additional-data** 存在，则 **additional-datum** 组件的个数将按照 C.21 的描述进行编码，且每一个 **additional-datum** 组件将按照下列两个小节的描述进行编码。

C.2.4.1 比特'0'（填充）被附加在比特流之后，且 **id** 组件按照 C.22 的描述进行编码。

C.2.4.2 比特'0'（填充）被附加在比特流之后，且 **data** 组件按照 C.22 的描述进行编码。

C.2.5 如果可选组件 **initial-vocabulary** 存在，则三个比特'000'（填充）将被附加在比特流之后，且该组件按照下列五个小节的描述进行编码。

C.2.5.1 对于 **initial-vocabulary**13 个可选组件（按照文本顺序）中的每一个，如果该组件存在，则比特'1'（表示存在）将被附加在比特流之后；否则比特'0'（表示缺失）将被附加在比特流之后。

C.2.5.2 如果 **initial-vocabulary** 中的可选组件 **external-vocabulary** 存在，则比特'0'（填充）将被附加在比特流之后，且该组件将按照 C.22 中的描述进行编码。

C.2.5.3 对于组件 **restricted-alphabets**，**encoding-algorithms**，**prefixes**，**namespace-names**，**local-names**，**other-ncnames** 和 **other-uris**（依此顺序）中的每一个，如果存在，则组件中的 **NonEmptyOctetString** 项的个数按照 C.21 的描述进行编码，然后每一项按照如下所述进行编码（按顺序）：比特'0'（填充）被附加在比特流之后，且 **NonEmptyOctetString** 按照 C.22 的描述进行编码。

C.2.5.4 对于组件 **attribute-values**，**content-character-chunks** 和 **other-strings**（依此顺序）中的每一个，如果存在，则组件中 **EncodedCharacterString** 项的个数按照 C.21 的描述进行编码，然后每一项按照如下所述进行编码（按顺序）：两个比特'00'（填充）被附加在比特流之后，且 **EncodedCharacterString** 按照 C.19 的描述进行编码。

C.2.5.5 对于组件 **element-name-surrogates** 和 **attribute-name-surrogates**（依此顺序）中的每一个，如果存在，则组件中的 **NameSurrogate** 项的个数按照 C.21 的描述进行编码，然后每一项按照如下所述进行编码（按顺序）：六个比特'000000'（填充）被附加在比特流之后，且 **NameSurrogate** 按照 C.16 的描述进行编码。

C.2.6 如果可选组件 **notations** 存在，则按照下面两小节的描述进行编码。

C.2.6.1 **notations** 中的每一项（按顺序）按照如下进行编码：六个比特'110000'（表示标识）被附加在比特流之后，且 **Notation** 按照 C.11 的描述进行编码。

C.2.6.2 四个比特'1111'（表示终止）以及四个比特'0000'（填充）被附加在比特流之后。

注 — 如果组件**notations**缺失，则不附加这些比特。

C.2.7 如果可选组件 **unparsed-entities** 存在，则按照下面两小节的描述进行编码。

C.2.7.1 **unparsed-entities** 中的每一项（按顺序）按照如下进行编码：七个比特'1101000'（表示标识）被附加在比特流之后，且 **UnparsedEntity** 按照 C.10 的描述进行编码。

C.2.7.2 四个比特'1111'（表示终止）以及四个比特'0000'（填充）被附加在比特流之后。

注 — 如果组件**unparsed-entities** 缺失，则不附加这些比特。

C.2.8 如果可选组件 **character-encoding-scheme** 存在，则比特'0'（填充）将被附加在比特流之后，且 **NonEmptyOctetString** 按照 C.22 的描述进行编码。

C.2.9 如果可选组件 **standalone** 存在，则按照如下所述进行编码：七个比特'0000000'（填充）被附加在比特流之后。如果 **standalone** 的取值为 **TRUE**，则比特'1'被附加在比特流之后，否则比特'0'被附加在比特流之后。

C.2.10 如果可选组件 **version** 存在，则其值按照 C.14 的描述进行编码。

C.2.11 如果组件 **children** 具有一个或多个项，则每一项按照下面五个小节的描述（按顺序）进行编码。

C.2.11.1 每一项的编码都要求起始于一个八位组的第一个比特。然而，附加的最近一个比特可能是一个八位组的第 8 个或第 4 个比特。如果是一个八位组的第 4 个比特，则'0000'（填充）将被附加在比特流之后，这样该项的编码就起始于下一个八位组的第一个比特了。

C.2.11.2 如果选择项 **element** 存在，则比特'0'（表示标识）被附加在比特流之后，且 **element** 按照 C.3 的描述进行编码。

C.2.11.3 如果选择项 **processing-instruction** 存在，则八个比特'11100001'（表示标识）被附加在比特流之后，且 **processing-instruction** 按照 C.5 的描述进行编码。

C.2.11.4 如果选择项 **comment** 存在，则八个比特'11100010'（表示标识）被附加在比特流之后，且 **comment** 按照 C.8 的描述进行编码。

C.2.11.5 如果选择项 **document-type-declaration** 存在，则六个比特'110001'（表示标识）被附加在比特流之后，且 **document-type-declaration** 按照 C.9 的描述进行编码。

C.2.12 四个比特'1111'（表示终止）被附加在比特流之后。

注 — 即使组件**children**没有项，也要附加这些比特。

C.3 Element 类型的编码

C.3.1 **Element** 类型在 7.3 节定义，如下所述：

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

C.3.2 **Element** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 此类型的一个编码总是起始于一个八位组的第二个比特，并且终止于另一个八位组的第四个或第八个比特（终止于C.3.8所描述的终止符'1111'的最后一个比特）。

C.3.3 如果可选组件 **attributes** 存在，则比特'1'（表示存在）被附加在比特流之后；否则比特'0'（表示缺失）被附加在比特流之后，

C.3.4 如果可选组件 **namespace-attributes** 存在，则按照下面三个小节的描述进行编码。

C.3.4.1 四个比特'1110'（表示存在）和两个比特'00'（填充）被附加在比特流之后。

C.3.4.2 **namespace-attributes** 中的每一项（按顺序）按照如下所述编码：六个比特'110011'（表示标识）被附加在比特流之后，且 **NamespaceAttribute** 按照 C.12 的描述进行编码。

C.3.4.3 四个比特'1111'（表示终止）和六个比特'000000'（填充）被附加在比特流之后。

注 — 如果组件**namespace-attributes** 缺失，则不附加这些比特。

C.3.5 组件 **qualified-name** 的值按照 C.18 的描述进行编码。

C.3.6 如果可选组件 **attributes** 存在，则按照下面两个小节的描述对它进行编码。

C.3.6.1 **attributes** 中的每一项（按顺序）按照如下所述进行编码：比特'0'（表示标识）被附加在比特流之后，且 **Attribute** 按照 C.4 的描述进行编码。

C.3.6.2 四个比特'1111'（表示终止）被附加在比特流之后。

注 — 如果组件**attributes**缺失，则不附加这些比特。

C.3.7 如果组件 **children** 具有一个或多个项，则每一项都按照下面六个小节的描述进行编码（按顺序）：

C.3.7.1 每一项的编码都要求起始于一个八位组的第一个比特。然而，附加的最近一个比特可能是一个八位组的第 8 个或第 4 个比特。如果是一个八位组的第 4 个比特，则'0000'（填充）将被附加在比特流之后，这样该项的编码就能起始于下一个八位组的第一个比特了。

C.3.7.2 如果选择项 **element** 存在，则比特'0'（表示标识）被附加在比特流之后，且 **element** 按照 C.3 的描述进行编码。

C.3.7.3 如果选择项 **processing-instruction** 存在，则八个比特'11100001'（表示标识）被附加在比特流之后，且 **processing-instruction** 按照 C.5 的描述进行编码。

C.3.7.4 如果选择项 **unexpanded-entity-reference** 存在，则六个比特'110010'（表示标识）被附加在比特流之后，且 **unexpanded-entity-reference** 按照 C.6 的描述进行编码。

C.3.7.5 如果选择项 **character-chunk** 存在，则两个比特'10'（表示标识）被附加在比特流之后，且 **character-chunk** 按照 C.7 的描述进行编码。

C.3.7.6 如果选择项 **comment** 存在，则八个比特'11100001'（表示标识）被附加在比特流之后，且 **comment** 按照 C.8 的描述进行编码。

C.3.8 四个比特'1111'（表示终止）被附加在比特流之后。

注 — 即使组件**children**没有项，也要附加这些比特。

C.4 Attribute 类型的编码

C.4.1 **Attribute** 类型在 7.4 节定义，如下所述：

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

C.4.2 **Attribute** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第二个比特，并且终止于另一个八位组的第八个比特。

C.4.3 **qualified-name** 的值按照 C.17 的描述进行编码。

C.4.4 **normalized-value** 的值按照 C.14 的描述进行编码。

C.5 ProcessingInstruction 类型的编码

C.5.1 **ProcessingInstruction** 类型在 7.5 节定义，如下所述：

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

C.5.2 **ProcessingInstruction** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第一个比特，并且终止于另一个八位组的第八个比特。

C.5.3 **target** 的值按照 C.13 的描述进行编码。

C.5.4 **content** 的值按照 C.14 的描述进行编码。

C.6 UnexpandedEntityReference 类型的编码

C.6.1 **UnexpandedEntityReference** 类型在 7.6 节定义，如下所述：

```
UnexpandedEntityReference ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                  -- OTHER NCNAME 类别 --,
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                  -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                  -- OTHER URI 类别 -- }
```

C.6.2 **UnexpandedEntityReference** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第七个比特，并且终止于另一个八位组的第八个比特。

C.6.3 对于每一个可选组件 **system-identifier** 和 **public-identifier**（按此顺序），如果该组件存在，则比特'1'（表示存在）将被附加在比特流之后，否则，比特'0'（表示缺失）将被附加在比特流之后。

C.6.4 **name** 的值按照 C.13 的描述进行编码。

C.6.5 如果可选组件 **system-identifier** 存在，则它按照 C.13 的描述进行编码。

C.6.6 如果可选组件 **public-identifier** 存在，则它按照 C.13 的描述进行编码。

C.7 CharacterChunk 类型的编码

C.7.1 **CharacterChunk** 类型在 7.7 节定义，如下所述：

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

C.7.2 **CharacterChunk** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第三个比特，并且终止于另一个八位组的第八个比特。

C.7.3 **character-codes** 的值按照 C.15 的描述进行编码。

C.8 Comment 类型的编码

C.8.1 **Comment** 类型在 7.8 节定义，如下所述：

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

C.8.2 **Comment** 类型的一个值通过执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第一个比特，并且终止于同一个八位组或另一个八位组的第八个比特。

C.8.3 **content** 的值按照 C.14 的描述进行编码。

C.9 DocumentTypeDeclaration 类型的编码

C.9.1 **DocumentTypeDeclaration** 类型在 7.9 节定义，如下所述：

```
DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                  -- OTHER URI 类别 --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                  -- OTHER URI 类别 --,
      children              SEQUENCE (SIZE(0..MAX)) OF
            ProcessingInstruction }
```

C.9.2 **DocumentTypeDeclaration** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第七个比特，并且终止于另一个八位组的第四个比特（是C.9.7中描述的终止符'1111'的最后一个比特）。

C.9.3 对于每一个可选组件 **system-identifier** 和 **public-identifier**（按此顺序），如果该组件存在，则比特'1'（表示存在）将被附加在比特流之后，否则，比特'0'（表示缺失）将被附加在比特流之后。

C.9.4 如果可选组件 **system-identifier** 存在，则它按照 C.13 的描述进行编码。

C.9.5 如果可选组件 **public-identifier** 存在，则它按照 C.13 的描述进行编码。

C.9.6 如果组件 **children** 具有一个或多个项，则每一项按照如下所述进行编码：八个比特'11100001'（表示标识）被附加在比特流之后，且 **ProcessingInstruction** 按照 C.5 的描述进行编码。

C.9.7 四个比特'1111'（表示终止）被附加在比特流之后。

注 — 即使组件**children**没有项，也要附加这些比特。

C.10 UnparsedEntity 类型的编码

C.10.1 **UnparsedEntity** 类型在 7.10 节定义，如下所述：

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

C.10.2 **UnparsedEntity** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第八个比特，并且终止于另一个八位组的第八个比特。

C.10.3 如果可选组件 **public-identifier** 存在，则比特'1'（表示存在）将被附加在比特流之后；否则，比特'0'（表示缺失）将被附加在比特流之后。

C.10.4 **name** 的值按照 C.13 的描述进行编码。

C.10.5 **system-identifier** 的值按照 C.13 的描述进行编码。

C.10.6 如果可选组件 **public-identifier** 存在，则它按照 C.13 的描述进行编码。

C.10.7 **name** 的值按照 C.13 的描述进行编码。

C.11 Notation 类型的编码

C.11.1 **Notation** 类型在 7.11 节定义，如下所述：

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

C.11.2 **Notation** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第八个比特，并且终止于另一个八位组的第八个比特。

C.11.3 对于每一个可选组件 **system-identifier** 和 **public-identifier**（按此顺序），如果该组件存在，则比特'1'（表示存在）将被附加在比特流之后；否则，比特'0'（表示缺失）将被附加在比特流之后。

C.11.4 **name** 的值按照 C.13 的描述进行编码。

C.11.5 如果可选组件 **system-identifier** 存在，则它按照 C.13 的描述进行编码。

C.11.6 如果可选组件 **public-identifier** 存在，则它按照 C.13 的描述进行编码。

C.12 NamespaceAttribute 类型的编码

C.12.1 **NamespaceAttribute** 类型在 7.12 节定义，如下所述：

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

C.12.2 **NamespaceAttribute** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第八个比特，并且终止于另一个八位组的第八个比特。

C.12.3 如果可选组件 **prefix** 存在，则比特'1'（表示存在）将被附加在比特流之后；否则，比特'0'（表示缺失）将被附加在比特流之后。

C.12.4 如果可选组件 **namespace-name** 存在，则比特'1'（表示存在）将被附加在比特流之后；否则，比特'0'（表示缺失）将被附加在比特流之后。

C.12.5 如果可选组件 **prefix** 存在，则它按照 C.13 的描述进行编码。

C.12.6 如果可选组件 **namespace-name** 存在，则它按照 C.13 的描述进行编码。

C.13 IdentifyingStringOrIndex类型的编码

C.13.1 **IdentifyingStringOrIndex** 类型在 7.13 节定义，如下所述：

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

C.13.2 **IdentifyingStringOrIndex** 类型的一个值通过执行如下动作来进行编码。

注 — 该类型的编码总是起始于一个八位组的第一个比特，并且终止于同一个八位组或另一个八位组的第八个比特。

C.13.3 如果选择项 **literal-character-string** 存在，则比特'0'（判别式）将被附加在比特流之后，且 **literal-character-string** 按照 C.22 的描述进行编码。

C.13.4 如果选择项 **string-index** 存在，则比特'1'（判别式）将被附加在比特流之后，且 **string-index** 按照 C.25 的描述进行编码。

C.14 起始于八位组第一个比特的 NonIdentifyingStringOrIndex 类型的编码

C.14.1 **NonIdentifyingStringOrIndex** 类型在 7.14 节定义，如下所述：

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

C.14.2 当编码是起始于一个八位组的第一个比特时（也见 C.15），将调用本小节 C.14 来对 **NonIdentifyingStringOrIndex** 类型的一个值进行编码。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于同一个八位组或另一个八位组的第八个比特。

C.14.3 如果选择项 **literal-character-string** 存在，则比特'0'（判别式）将被附加在比特流之后，且 **literal-character-string** 按照下列两个小节的描述进行编码。

C.14.3.1 如果组件 **add-to-table** 取值为 **TRUE**，则比特'1'将被附加在比特流之后；否则比特'0'将被附加在比特流之后。

C.14.3.2 组件 **character-string** 的值按照 C.19 的描述进行编码。

C.14.4 如果选择项 **string-index** 存在，则比特'1'（判别式）将被附加在比特流之后，且 **string-index** 按照 C.26 的描述进行编码。

C.15 起始于八位组第三个比特的 NonIdentifyingStringOrIndex 类型的编码

C.15.1 **NonIdentifyingStringOrIndex** 类型在 7.14 节定义，如下所述：

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

C.15.2 当编码是起始于一个八位组的第三个比特时（也见 C.14），将调用本小节 C.15 来对 **NonIdentifyingStringOrIndex** 类型的一个值进行编码。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于同一个八位组或另一个八位组的第八个比特。

C.15.3 如果选择项 **literal-character-string** 存在，则比特'0'（判别式）将被附加在比特流之后，且 **literal-character-string** 按照下列两个小节的描述进行编码。

C.15.3.1 如果组件 **add-to-table** 取值为 **TRUE**，则比特'1'将被附加在比特流之后；否则比特'0'将被附加在比特流之后。

C.15.3.2 组件 **character-string** 的值按照 C.20 的描述进行编码。

C.15.4 如果选择项 **string-index** 存在，则比特'1'（判别式）将被附加在比特流之后，且 **string-index** 按照 C.28 的描述进行编码。

C.16 NameSurrogate 类型的编码

C.16.1 **NameSurrogate** 类型在 7.15 节定义，如下所述：

```
NameSurrogate ::= SEQUENCE {
     prefix-string-index           INTEGER(1..one-meg) OPTIONAL,
     namespace-name-string-index   INTEGER(1..one-meg) OPTIONAL,
     local-name-string-index       INTEGER(1..one-meg) }
     (CONSTRAINED BY  {--当且仅当 namespace-name-string-index 存在时,
                      -- prefix-string-index 才存在 --})
```

C.16.2 **NameSurrogate** 类型的一个值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是起始于异构八位组的第七个比特，并终止于另一个八位组的第八个比特。

C.16.3 如果可选组件 **prefix-string-index** 存在，则比特'1'（表示存在）将被附加在比特流之后；否则，比特'0'（表示缺失）将被附加在比特流之后。

C.16.4 如果可选组件 **namespace-name-string-index** 存在，则比特'1'（表示存在）将被附加在比特流之后；否则，比特'0'（表示缺失）将被附加在比特流之后。

C.16.5 如果可选组件 **prefix-string-index** 存在，则比特'0'（填充）将被附加在比特流之后，且该组件按照 C.25 的描述进行编码。

C.16.6 如果可选组件 **namespace-name-string-index** 存在，则比特'0'（填充）将被附加在比特流之后，且该组件按照 C.25 的描述进行编码。

C.16.7 比特'0'（填充）被附加在比特流之后，且组件 **local-name-string-index** 按照 C.25 的描述进行编码。

C.17 起始于八位组第二个比特的 QualifiedNameOrIndex 类型的编码

C.17.1 **QualifiedNameOrIndex** 类型在 7.16 节定义，如下所述：

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

C.17.2 当编码是起始于一个八位组的第二个比特时（也见 C.18），将调用本小节 C.17 来对 **QualifiedNameOrIndex** 类型的一个值进行编码。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于同一个八位组或另一个八位组的第八个比特。

C.17.3 如果选择项 **literal-qualified-name** 存在，则四个比特'1111'（表示标识）和比特'0'（填充）将被附加在比特流之后，且 **literal-qualified-name** 按照下面四个小节的描述进行编码。

C.17.3.1 对于每一个可选组件 **prefix** 和 **namespace-name**（按此顺序），如果该组件存在，则比特'1'（表示存在）将被附加在比特流之后；否则比特'0'（表示缺失）将被附加在比特流之后。

C.17.3.2 如果可选组件 **prefix** 存在，则它按照 C.13 的描述进行编码。

C.17.3.3 如果可选组件 **namespace-name** 存在，则它按照 C.13 的描述进行编码。

C.17.3.4 组件 **local-name** 按照 C.13 的描述进行编码。

C.17.4 如果选择项 **name-surrogate-index** 存在，则它按照 C.25 的描述进行编码。

C.18 起始于八位组第三个比特的 QualifiedNameOrIndex 类型的编码

C.18.1 **QualifiedNameOrIndex** 类型在 7.16 节定义，如下所述：

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

C.18.2 当编码是起始于一个八位组的第三个比特时（也见 C.17），将调用本小节 C.18 来对 **QualifiedNameOrIndex** 类型的一个值进行编码。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于同一个八位组或另一个八位组的第八个比特。

C.18.3 如果选择项 **literal-qualified-name** 存在，则四个比特'1111'（表示标识）将被附加在比特流之后，且 **literal-qualified-name** 按照下面四个小节的描述进行编码。

C.18.3.1 对于每一个可选组件 **prefix** 和 **namespace-name**（按此顺序），如果该组件存在，则比特'1'（表示存在）将被附加在比特流之后；否则比特'0'（表示缺失）将被附加在比特流之后。

C.18.3.2 如果可选组件 **prefix** 存在，则它按照 C.13 的描述进行编码。

C.18.3.3 如果可选组件 **namespace-name** 存在，则它按照 C.13 的描述进行编码。

C.18.3.4 组件 **local-name** 按照 C.13 的描述进行编码。

C.18.4 如果选择项 **name-surrogate-index** 存在，则它按照 C.27 的描述进行编码。

C.19 起始于八位组第三个比特的 EncodedCharacterString 类型的编码

C.19.1 **EncodedCharacterString** type 类型在 7.17 节定义，如下所述：

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

C.19.2 当编码是起始于一个八位组的第三个比特时（也见 C.20），将调用本小节 C.19 来对 **EncodedCharacterString** 类型的一个值进行编码。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于另一个八位组的第八个比特。

C.19.3 组件 **encoding-format** 的值将按照下面四个小节的描述进行编码。

C.19.3.1 如果选择项 **utf-8** 存在，则两个比特'00'（判别式）将被附加在比特流之后。

C.19.3.2 如果选择项 **utf-16** 存在，则两个比特'01'（判别式）将被附加在比特流之后。

C.19.3.3 如果选择项 **restricted-alphabet** 存在，则两个比特'10'（判别式）将被附加在比特流之后，且 **restricted-alphabet** 按照 C.29 的描述进行编码。

C.19.3.4 如果选择项 **encoding-algorithm** 存在，则两个比特'11'（判别式）将被附加在比特流之后，且 **encoding-algorithm** 按照 C.29 的描述进行编码。

C.19.4 组件 **octets** 按照 C.23 的描述进行编码。

C.20 起始于八位组第五个比特的 EncodedCharacterString 类型的编码

C.20.1 **EncodedCharacterString** 类型在 7.17 节定义，如下所述：

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

C.20.2 当编码是起始于一个八位组的第五个比特时（也见 C.19），将调用本小节 C.20 来对 **EncodedCharacterString** 类型的一个值进行编码。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于另一个八位组的第八个比特。

C.20.3 组件 **encoding-format** 的值将按照下面四个小节的描述进行编码。

C.20.3.1 如果选择项 **utf-8** 存在，则两个比特'00'（判别式）将被附加在比特流之后。

C.20.3.2 如果选择项 **utf-16** 存在，则两个比特'01'（判别式）将被附加在比特流之后。

C.20.3.3 如果选择项 **restricted-alphabet** 存在，则两个比特'10'（判别式）将被附加在比特流之后，且 **restricted-alphabet** 按照 C.29 的描述进行编码。

C.20.3.4 如果选择项 **encoding-algorithm** 存在，则两个比特'11'（判别式）将被附加在比特流之后，且 **encoding-algorithm** 按照 C.29 的描述进行编码。

C.20.4 组件 **octets** 按照 C.24 的描述进行编码。

C.21 sequence-of 类型的长度的编码

C.21.1 在对一个 sequence-of 类型编码时，如果在 sequence-of 类型的项之前有一个长度字段，则调用本小节来对一个 sequence-of 类型的长度进行编码。

注 — 这种编码总是起始于一个八位组的第一个比特，并且终止于同一个八位组或另一个八位组的第八个比特。

C.21.2 如果长度值在 1 到 128 的范围之内，则在比特流之后附加一个比特'0'，且其值在减去此范围的低界后，被编码为一个无符号整数，放在七个比特的字段中，并附加在比特流之后。

C.21.3 如果长度值在 129 到 2^{20} 的范围之内，则在比特流之后附加一个比特'1'和三个比特'000'（填充），且其值在减去此范围的低界后，被编码为一个无符号整数，放在二十个比特的字段中，并附加在比特流之后。

C.22 起始于八位组第二个比特的NonEmptyOctetString 类型的编码

C.22.1 **NonEmptyOctetString** 类型在 7.2 节定义，如下所述：

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

C.22.2 当编码是起始于一个八位组的第二个比特时，调用本小节 C.22 来对 **NonEmptyOctetString** 类型的一个值进行编码（也见 C.23 和 C.24）。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于另一个八位组的第八个比特。

C.22.3 八位组串的长度按照下面三个小节的描述进行编码。

C.22.3.1 如果长度值在 1 到 64 的范围之内，则在比特流之后附加一个比特'0'，且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个六比特的字段中，并附加在比特流之后。

C.22.3.2 如果长度值在 65 到 320 的范围之内，则在比特流之后附加两个比特'10'和五个比特'00000'（填充），且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个八比特的字段中，并附加在比特流之后。

C.22.3.3 如果长度值在 321 到 2^{32} 的范围之内，则在比特流之后附加两个比特'11'和五个比特'00000'（填充），且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个三十二比特的字段中，并附加在比特流之后。

C.22.4 构成八位组串中八位组的比特被附加在比特流之后（按顺序）。

C.23 起始于八位组第五个比特的 NonEmptyOctetString 类型的编码

C.23.1 **NonEmptyOctetString** 类型在 7.2 节定义，如下所述：

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

C.23.2 当编码是起始于一个八位组的第五个比特时，调用本小节 C.23 来对 **NonEmptyOctetString** 类型的一个值进行编码（也见 C.22 和 C.24）。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于另一个八位组的第八个比特。

C.23.3 八位组串的长度按照下面三个小节的描述进行编码。

C.23.3.1 如果长度值在 1 到 8 的范围之内，则在比特流之后附加一个比特'0'，且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个三比特的字段中，并附加在比特流之后。

C.23.3.2 如果长度值在 9 到 264 的范围之内，则在比特流之后附加两个比特'10'和两个比特'00'（填充），且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个八比特的字段中，并附加在比特流之后。

C.23.3.3 如果长度值在 265 到 2^{32} 的范围之内，则在比特流之后附加两个比特'11'和两个比特'00'（填充），且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个三十二比特的字段中，并附加在比特流之后。

C.23.4 构成八位组串中八位组的比特被附加在比特流之后（按顺序）。

C.24 起始于八位组第七个比特的NonEmptyOctetString 类型的编码

C.24.1 **NonEmptyOctetString** 类型在 7.2 节定义，如下所述：

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

C.24.2 当编码是起始于一个八位组的第七个比特时，调用本小节 C.24 来对 **NonEmptyOctetString** 类型的一个值进行编码（也见 C.22 和 C.23）。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于另一个八位组的第八个比特。

C.24.3 八位组串的长度按照下面三个小节的描述进行编码。

C.24.3.1 如果长度值在 1 到 2 的范围之内，则在比特流之后附加一个比特'0'，且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个比特的字段中，并附加在比特流之后。

C.24.3.2 如果长度值在 3 到 258 的范围之内，则在比特流之后附加两个比特'10'，且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个八比特的字段中，并附加在比特流之后。

C.24.3.3 如果长度值在 259 到 2^{32} 的范围之内，则在比特流之后附加两个比特'11'，且该长度值在减去此范围的低界后，被编码为一个无符号整数，放在一个三十二比特的字段中，并附加在比特流之后。

C.24.4 构成八位组串中八位组的比特被附加在比特流之后（按顺序）。

C.25 起始于八位组第二个比特的1 到2^{20} 范围内整数的编码

C.25.1 当编码是起始于一个八位组的第二个比特时，调用本小节 C.25 来对一个范围在 1 到 2^{20} 的整数值进行编码（也见 C.26，C.27 和 C.28）。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于同一个八位组或另一个八位组的第八个比特。

C.25.2 如果该值在 1 到 64 的范围之内，则在比特流之后附加一个比特'0'，且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个六比特的字段中，并附加在比特流之后。

C.25.3 如果该值在 65 到 8256 的范围之内，则在比特流之后附加两个比特'10'，且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个十三比特的字段中，并附加在比特流之后。

C.25.4 如果该值在 8257 到 2^{20} 的范围之内，则在比特流之后附加两个比特'11'和一个比特'0'（填充），且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个二十比特的字段中，并附加在比特流之后。

C.26 起始于八位组第二个比特的0 到2^{20} 范围内整数的编码

C.26.1 当编码是起始于一个八位组的第二个比特时，调用本小节 C.26 来对一个范围在 0 到 2^{20} 的整数值进行编码（也见 C.25，C.27 和 C.28）。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于同一个八位组或另一个八位组的第八个比特。

C.26.2 如果该值为零，则在比特流之后附加七个比特'1111111'。否则该值按照 C.25 的描述进行编码。

C.27 起始于八位组第三个比特的1 到2^{20} 范围内整数的编码

C.27.1 当编码是起始于一个八位组的第三个比特时，调用本小节 C.27 来对一个范围在 1 到 2^{20} 的整数值进行编码（也见 C.25，C.26 和 C.28）。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于同一个八位组或另一个八位组的第八个比特。

C.27.2 如果该值在 1 到 32 的范围之内，则在比特流之后附加一个比特'0'，且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个五比特的字段中，并附加在比特流之后。

C.27.3 如果该值在 33 到 2080 的范围之内，则在比特流之后附加三个比特'100'，且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个十一比特的字段中，并附加在比特流之后。

C.27.4 如果该值在 2081 到 526368 的范围之内，则在比特流之后附加三个比特'101'，且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个十九比特的字段中，并附加在比特流之后。

C.27.5 如果该值在 526369 到 2^{20} 的范围之内，则在比特流之后附加三个比特'110"和七个比特'0000000'（填充），且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个二十比特的字段中，并附加在比特流之后。

C.28 起始于八位组第四个比特的1 到2^{20} 范围内整数的编码

C.28.1 当编码是起始于一个八位组的第四个比特时，调用本小节 C.28 来对一个范围在 1 到 2^{20} 的整数值进行编码（也见 C.25，C.26 和 C.27）。该值通过（按顺序）执行如下动作来进行编码。

注 — 该类型的编码总是终止于同一个八位组或另一个八位组的第八个比特。

C.28.2 如果该值在 1 到 16 的范围之内，则在比特流之后附加一个比特'0'，且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个四比特的字段中，并附加在比特流之后。

C.28.3 如果该值在 17 到 1040 的范围之内，则在比特流之后附加三个比特'100'，且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个十比特的字段中，并附加在比特流之后。

C.28.4 如果该值在 1041 到 263184 的范围之内，则在比特流之后附加三个比特'101'，且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个十八比特的字段中，并附加在比特流之后。

C.28.5 如果该值在 263185 到 2^{20} 的范围之内，则在比特流之后附加三个比特'110"和六个比特'000000'（填充），且该值在减去此范围的低界后，被编码为一个无符号整数，放在一个二十比特的字段中，并附加在比特流之后。

C.29 1 到256范围内整数的编码

C.29.1 调用本小节 C.29 来对一个范围在 1 到 256 的整数值进行编码。

注 — 该类型的编码总是起始于一个八位组的第五个或第七个比特，并（分别）终止于下一个八位组的第四个或第六个比特。

C.29.2 该值减去此范围的低界后，被编码为一个无符号整数，放在一个八比特的字段中，并附加在比特流之后。

附 件 D

将XML信息集编码为快速信息集文档的示例

（本附件不是本建议书 | 国际标准的组成部分）

D.1 示例介绍

D.1.1 本附件针对数字使用下列印刷约定：

a） 对于一个以10进制表示的数，对数中的阿拉伯数字使用**粗体**`Courier`来表示，并跟随一个下标"10"（例如，$\texttt{11}_{10}$）；且

b） 对于一个以16进制表示的数（一个十六进制数），对数中的数字使用**粗体**`Courier`来表示，并跟随一个下标"16"（例如，$\texttt{0b1f}_{16}$）；且

c） 如果数的基数被显式地声明，则下标可以忽略。

D.1.2 本附件给出了将一个通用商务语言（UBL）[1]定单编码为一个快速信息集文档的两种可能的编码示例。UBL 是被设计用来提供一种能够被通用理解的和公认的商业语法，以便用来合法地绑定商务文档。

D.1.3 针对示例 UBL 定单的 XML 信息集在 D.3 中给出。

D.1.4 第一种快速信息集文档具有一个引用了某个外部词汇的初始词汇。D.4 小节描述了外部词汇的内容，快速信息集文档中的八位组，以及对某些八位组序列的解释。

D.1.5 第二种快速信息集文档没有初始词汇。D.5 小节描述了快速信息集文档中的八位组，以及对某些八位组序列的解释。

注 — 该快速信息集文档的最终词汇与D.4中描述的快速信息集文档的最终词汇相同。

D.1.6 D.4 和 D.5 小节中的八位组通过一系列表格给出，每个表格都具有两列。第一列列出了快速信息集文档中的 32 个连续八位组的起始位置，以 16 进制表示，第二列列出了以 16 进制表示的这些八位组。包含了对应于标识信息项和终止信息项的比特的那些 16 进制字符下面增加了下划线。

D.1.7 针对快速信息集文档中的某些八位组序列（在 D.4 和 D.5 中）的解释在表格中给出，并具有下面的列：

a） 第1列给出了第2列中列出的八位组的位置，以16进制表示。

b） 第2列给出了与相应信息项和信息项属性相关联的快速信息集文档中的八位组。一个八位组以二进制给出，并在后面跟一个同一八位组的16进制表示，放在括号内，例如`11110000 (f0)`。

c） 第3列给出了对第2列中八位组的详细描述，并且引用了附件C中的小节作更进一步的解释和澄清。

d） 第4列给出了与第2列中的八位组相对应的XML信息集的一部分或者XML 1.0文档的一部分（如果可应用的话）。

D.1.8 在这些示例中，所有包含了少于 6 个字符的 **character** 信息项的块都被加入到 CONTENT CHARACTER CHUNK 表中，且所有包含了少于 6 个字符的 **attribute** 信息项的**[normalized value]** 属性都被加入到 ATTRIBUTE VALUE 表中。

D.1.9 XML 1.0 文档和快速信息集文档的长度，以及对这些文档压缩后的长度（使用 GZIP）都列在 D.2 中。

D.2 示例文档的长度（包括基于冗余的压缩）

D.2.1 表 D.1 给出了所有文档的长度。第 1 列列出了 UBL 文档，第 2 列列出了文档长度，第 3 列列出了文档经 GZIP（采用缺省选项）压缩后的长度。

注 1 — UBL定单XML 1.0文档中没有包含空白字符（见D.3.1.2）。

注 2 — 对于每一个文档，所有的字符都使用UTF-8字符编码进行编码。

注 3 — 针对此快速信息集文档，没有XML声明（见12.3）。

表 D.1－文档的初始长度和GZIP压缩后的长度

UBL文档	长　　度	GZIP压缩长度
XML 1.0 文档	3311	893
具有一个外部词汇的快速信息集文档	684	546
没有初始词汇的快速信息集文档	1322	860

D.2.2 在长度比较中，具有一个到外部词汇的引用的快速信息集文档的长度最小，在经 GZIP 压缩后的长度比较中也是最小。经 GZIP 压缩后的长度与快速信息集文档长度之间的比率意味着此快速信息集文档具有非常少的冗余信息。

D.2.3 在所有情况下，快速信息集文档经 GZIP 压缩后的长度都小于 XML 1.0 文档经 GZIP 压缩后的长度。此外，具有一个到外部词汇的引用的快速信息集文档的长度小于 XML 1.0 文档经 GZIP 压缩后的长度。

D.3 UBL定单示例

D.3.1 木制品定单示例

D.3.1.1 UBL 定单示例取自[1]。特别地，选择木制品定单示例（见 xml/joinery/UBL-Order-1.0-Joinery-Example.xml）是基于如下原因：

a） 这是一个现实世界中的例子，其开发是独立于本建议书 | 国际标准的，且对于快速信息集没有特殊的偏向；

b） 它是随意可用的；且

c） 它广泛地使用了XML命名空间，并因此是一个很好的示例来表明快速信息集是如何支持XML命名空间的。

D.3.1.2 木制品定单示例作了如下修改：

a） 最后三个**OrderLine** 元素被删除；且

注 1 — 这使得XML 1.0文档被缩减到一个合理的长度，可以在本建议书 | 国际标准中表示。

b） 所有的空白字符都被删除。

注 2 — 这表示了XML信息集的一种更现实的使用用例，可以被系列化、被跨网传输，被解析等。

D.3.2 木制品定单XML 1.0文档

木制品定单 XML 1.0 文档做了如 D.3.1.2 a 所声明的修改，但为了可读性，空白字符还是保留了，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:res="urn:oasis:names:tc:ubl:codelist:AcknowledgementResponseCode:1:0"
xmlns:cbc="urn:oasis:names:tc:ubl:CommonBasicComponents:1:0"
xmlns:cac="urn:oasis:names:tc:ubl:CommonAggregateComponents:1:0"
xmlns:cur="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oasis:names:tc:ubl:Order:1:0"
xsi:schemaLocation="urn:oasis:names:tc:ubl:Order:1:0../../xsd/maindoc/UBL-Order-1.0.xsd">
        <BuyersID>S03-034257</BuyersID>
        <cbc:IssueDate>2003-02-03</cbc:IssueDate>
        <cac:BuyerParty>
                <cac:Party>
                        <cac:PartyName>
                                <cbc:Name>Jerry Builder plc</cbc:Name>
                        </cac:PartyName>
                        <cac:Address>
                                <cbc:StreetName>Marsh Lane</cbc:StreetName>
                                <cbc:CityName>Nowhere</cbc:CityName>
                                <cbc:PostalZone>NR18 4XX</cbc:PostalZone>
                                <cbc:CountrySubentity>Norfolk</cbc:CountrySubentity>
                        </cac:Address>
                        <cac:Contact>
                                <cbc:Name>Eva Brick</cbc:Name>
                        </cac:Contact>
                </cac:Party>
        </cac:BuyerParty>
        <cac:SellerParty>
```

```
                <cac:Party>
                    <cac:PartyName>
                        <cbc:Name>Specialist Windows plc</cbc:Name>
                    </cac:PartyName>
                    <cac:Address>
                        <cbc:BuildingName>Snowhill Works</cbc:BuildingName>
                        <cbc:CityName>Little Snoring</cbc:CityName>
                        <cbc:PostalZone>SM2 3NW</cbc:PostalZone>
                        <cbc:CountrySubentity>Whereshire</cbc:CountrySubentity>
                    </cac:Address>
                </cac:Party>
            </cac:SellerParty>
            <cac:Delivery>
                <cbc:RequestedDeliveryDateTime>2003-02-24T00:00:00</cbc:RequestedDeliveryDateTime>
                <cac:DeliveryAddress>
                    <cbc:StreetName>Riverside Rd.</cbc:StreetName>
                    <cbc:BuildingName>Plot 17, Whitewater Estate</cbc:BuildingName>
                    <cbc:CityName>Whetstone</cbc:CityName>
                    <cbc:CountrySubentity>Middlesex</cbc:CountrySubentity>
                </cac:DeliveryAddress>
            </cac:Delivery>
            <cac:OrderLine>
                <cac:LineItem>
                    <cac:BuyersID>A</cac:BuyersID>
                    <cbc:Quantity quantityUnitCode="unit">2</cbc:Quantity>
                    <cac:Item>
                        <cac:SellersItemIdentification>
                            <cac:ID>236WV</cac:ID>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>wood</cac:AttributeID>
                                <cbc:Description>soft</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>finish</cac:AttributeID>
                                <cbc:Description>primed</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>fittings</cac:AttributeID>
                                <cbc:Description>satin</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>glazing</cac:AttributeID>
                                <cbc:Description>single</cbc:Description>
                            </cac:PhysicalAttribute>
                        </cac:SellersItemIdentification>
                    </cac:Item>
                </cac:LineItem>
            </cac:OrderLine>
            <cac:OrderLine>
                <cac:LineItem>
                    <cac:BuyersID>B</cac:BuyersID>
                    <cbc:Quantity quantityUnitCode="unit">3</cbc:Quantity>
                    <cac:Item>
                        <cac:SellersItemIdentification>
                            <cac:ID>340TW</cac:ID>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>hand</cac:AttributeID>
                                <cbc:Description>RH</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>wood</cac:AttributeID>
                                <cbc:Description>hard</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>finish</cac:AttributeID>
                                <cbc:Description>stain</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>fittings</cac:AttributeID>
                                <cbc:Description>brass</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>glazing</cac:AttributeID>
                                <cbc:Description>double</cbc:Description>
                            </cac:PhysicalAttribute>
```

```
                    </cac:SellersItemIdentification>
                </cac:Item>
            </cac:LineItem>
        </cac:OrderLine>
    </Order>
```

D.4 具备外部词汇的UBL定单快速信息集文档

快速信息集文档的外部词汇在 D.4.1 中给出。快速信息集文档的八位组（作为 16 进制字符）在 D.4.2 中给出。关于 D.4.2 中某些八位组序列的详细解释在 D.4.3 中给出。此快速信息集文档不能被认为是自我描述的，因为需要外部信息（外部词汇）来产生完整的 XML 信息集。

注 — 如果一个快速信息集的解析器不能通过给定的URI获取到词汇表，则快速信息集文档还是可以被该解析器处理，但是不能通过对词汇表索引的解引用来获取到产生信息项属性的必要信息。

D.4.1 UBL定单的外部词汇

D.4.1.1 快速信息集文档的外部词汇被规定为是从示例 UBL 定单 XML 信息集（见 D.3.1.2）中获取到的最终词汇，而该信息集被进一步修改，以便包含：

a） 没有**character** 信息项；且

b） **attribute** 信息项中有空的**[normalized value]**属性。

注 1 — 这表示一种实际的场景，在此场景中，一个XML信息集的应用所定义的内容（**character**信息项和/或**attribute**信息项的**[normalized value]**属性）将是什么并不能提前知晓。

注 2 — 实际上，并不期望使用序列化的文档来产生外部词汇。可以预见的是工具将利用模式，以及模式的潜在的XML信息集实例来对串和限定名字的出现频率进行分析，这样更小的索引值将会分配给更频繁出现的信息（例如，XML信息集中**[local name]**属性的出现频率可能服从一种幂次级数）。

D.4.1.2 外部词汇的 URI 为：**urn:oasis:names:tc:ubl:Order:1.0:joinery:example**。

D.4.1.3 表 D.2 给出了 UBL 定单 XML 信息集的词汇。第 1 列列出了词汇表的词汇表索引（索引），第 2 列列出了 PREFIX 表的词汇表条目（前缀表条目），第 3 列列出了 NAMESPACE NAME 表的词汇表条目（命名空间名字条目），第 4 列列出了 LOCAL NAME 表的词汇表条目（本地名字条目），第 5 列列出了 ELEMENT NAME 表的词汇表条目（元素名字条目），第 6 列列出了 ATTRIBUTE NAME 表的词汇表条目（属性名字条目）。对于 ELEMENT NAME 表和 ATTRIBUTE NAME 表中的名字代理条目的索引值，按照为 **`NameSurrogate`** 类型的组件所规定的顺序给出（即 **`prefix-name-string-index`**，**`namespace-name-string-index`** 和 **`local-name-string-index`**）。一个字符"_"表示该值缺失（这仅可能在 **`prefix-name-string-index`** 和 **`namespace-name-string-index`** 组件的值中出现）。

注 1 — 对应于XML前缀为"xml"，以及XML命名空间名字为"http://www.w3.org/XML/1998/namespace"的前缀和命名空间名字的第一个条目（索引为1）是内置的（见7.2.21节和7.2.22节）。

注 2 — 长的命名空间名字条目（URI）已经被截短了。

注 3 — 对于第一个元素名字条目（索引为1），没有指向一个前缀的引用（由于该值缺失，表示为"_"），有一个引用为**[namespace name]**属性（"urn:oasis:names:tc:ubl:Order:1:0"）指向了第七个命名空间名字条目（索引为7），并且有一个引用为**[local name]**属性（"Order"）指向了第一个本地名字条目（索引为1）。

表 D.2－UBL定单XML信息集的词汇

索引	前缀条目	命名空间名字条目	本地名字条目	元素名字条目	属性名字条目
1	xml	http://www.w3.org/XML/1998/namespace	Order	_ 7 1	6 6 2
2	res	….AcknowledgementResponseCode:1:0	schemaLocation	_ 7 3	_ _ 23
3	cbc	….CommonBasicComponents:1:0	BuyersID	3 3 4	
4	cac	….CommonAggregateComponents:1:0	IssueDate	4 4 5	
5	cur	….CurrencyCode:1:0	BuyerParty	4 4 6	
6	xsi	….XMLSchema-instance	Party	4 4 7	
7		….Order:1:0	PartyName	3 3 8	
8			Name	4 4 9	
9			Address	3 3 10	
10			StreetName	3 3 11	
11			CityName	3 3 12	
12			PostalZone	3 3 13	
13			CountrySubentity	4 4 14	
14			Contact	4 4 15	
15			SellerParty	3 3 16	
16			BuildingName	4 4 17	
17			Delivery	3 3 18	
18			RequestedDeliveryDateTime	4 4 19	
19			DeliveryAddress	4 4 20	
20			OrderLine	4 4 21	
21			LineItem	4 4 3	
22			Quantity	3 3 22	
23			quantityUnitCode	4 4 24	
24			Item	4 4 25	
25			SellersItemIdentification	4 4 26	
26			ID	4 4 27	
27			PhysicalAttribute	4 4 28	
28			AttributeID	3 3 29	
29			Description		

D.4.2 快速信息集文档的八位组（16进制字符）

表 D.3 给出了 D.3 中给出的 UBL 定单示例的快速信息集文档的八位组。

注 — 包含了对应于信息项的标识符和终止符比特的那些16进制字符用下划线标识。

表 D.3－快速信息集文档的八位组（16进制字符）

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000000	e00100002010002f75726e3a6f617369733a6e616d65733a74633a75626c3a4f
000020	726465723a313a303a6a6f696e6572793a6578616d706c6578cf8181cf8282cf
000040	8383cf8484cf8585cd86f00000083b75726e3a6f617369733a6e616d65733a74
000060	633a75626c3a4f726465723a313a30202e2e2f2e2e2f7873642f6d61696e646f
000080	632f55424c2d4f726465722d312e302e787364f00182075330332d3033343235
0000a0	37f00282073230332d30322d3033f00304050682 0e4a6572727920427 5696c
0000c0	64657220706c63ff070882074d61727368204c616e65f00982044e6f77686572
0000e0	65f00a82054e52313820345858f00b82044e6f72666f6c6bff0c068206457661
000100	20427269636bffff0d04050682135370656369616c6973742057696e646f7773
000120	20706c63ff070e820b536e6f7768696c6c20576f726b73f009820b4c6974746c
000140	6520536e6f72696e67f00a8204534d3220334e57f00b820757686572657265736869
000160	7265ffff0f108210323030332d30322d32345430303a30303a3030f01108820a
000180	526976657273696464652052642ef00e8217506c6f742031372c20576869746577
0001a0	617465722045737461746 5f00982065768657473746f6e65f00b82064d696464
0001c0	6c65736578fff01213149041f0550143756e6974f09032f01617189202323336
0001e0	5756f0191a9201776f6f64f01b9201736f6674ff191a820366696e697368f01b
000200	82037072696d6564ff191a8205666974746 96e6773f01b9202736174696eff19
000220	1a8204676c617a696e67f01b820373696e676c65ffffff1213149042f0550180
000240	f09033f016171892023334305457f0191a920168616e64f01b915248ff191aa3
000260	f01b920168617264ff191a820366696e697368f01b9202737461696eff191a82
000280	0566697474696e6773f01b9202627261737 3ff191a8204676c617a696e67f01b
0002a0	8203646f75626c65ffffffff
0002ac	

D.4.3 编码的解释

D.4.3.1 document 信息项和Order元素信息项的编码

下列解释详细说明了快速信息集文档（包括外部词汇的 URI）和根元素信息项的初始编码。特别的，对一个 **document** 信息项，一系列命名空间（**namespace**）信息项，一个元素（**element**）信息项，以及一个属性（**attribute**）信息项的编码进行了解释。表 D.4 给出了对 D.3.2 中 **document** 信息项和 **Order** 元素信息项进行编码的快速信息集文档的片段。表 D.5 详细解释了此编码。以 XML 1.0 表示的片段如下所示：

```
<Order xmlns:res="urn:oasis:names:tc:ubl:codelist:AcknowledgementResponseCode:1:0"
xmlns:cbc="urn:oasis:names:tc:ubl:CommonBasicComponents:1:0"
xmlns:cac="urn:oasis:names:tc:ubl:CommonAggregateComponents:1:0"
xmlns:cur="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oasis:names:tc:ubl:Order:1:0"
xsi:schemaLocation="urn:oasis:names:tc:ubl:Order:1:0 ../../xsd/maindoc/UBL-Order-1.0.xsd">
```

表 D.4－八位组片段（16进制字符）

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000000	e00100002010002f75726e3a6f617369733a6e616d65733a74633a75626c3a4f
000020	726465723a313a303a6a6f696e6572793a6578616d706c6578cf8181cf8282cf
000040	8383cf8484cf8585cd86f00000083b75726e3a6f617369733a6e616d65733a74
000060	633a75626c3a4f726465723a313a30202e2e2f2e2e2f7873642f6d61696e646f
000080	632f55424c2d4f726465722d312e302e787364f0

表 D.5－编码详述

	八 位 组	描 述	XML信息集或XML
00 01	11100000 (e0) 00000000 (00)	这些八位组应出现在每个快速信息集文档的起始位置（见 12.6）。	**document** 信息项
02 03	00000000 (00) 00000001 (01)	这些八位组为版本号的编码（见 12.9）。	
04 05 06	00100000 (20) 00010000 (10) 00000000 (00)	这些八位组的编码表示存在一个初始词汇表，且有一个引用指向了此初始词汇的一个外部词汇。 所处位置为 04_{16} 的八位组，其值为 20_{16}，其中第一个比特为一个'0'（填充）（见 12.8 节）。第三个比特为'1'，表示 **initial-vocabulary** 组件存在，且其他六个可选组件都缺失（见 C.2.3）。 所处位置为 05_{16} 的八位组，其值为 10_{16}，其中前三个比特为三个'0'（填充）（见 C.2.5）。第四个比特为'1'，表示 **initial-vocabulary** 中的 **external-vocabulary** 存在。最后四个比特为'0'（第五个到第八个比特），表示十二个其他可选组件中的四个都缺失（见 C.2.5.1）。 所处位置为 06_{16} 的八位组，其值为 00_{16}，所有的比特都是'0'，表示十二个可选组件中的最后八个都缺失（见 C.2.5.1）。	
07 08 37	00101111 (2f) 01110101 (75) 01100101 (65)	这些八位组是对外部词汇的 URI 的编码。 所处位置为 07_{16} 的八位组，其值为 $2f_{16}$，其中第一个比特为一个'0'（填充）。URI 被编码为 UTF-8 字符（见 C.22）。第二个比特为'0'，表示 URI 的长度大于或等于 1_{10} 个八位组，且小于或等于 64_{10} 个八位组，同时该长度减去此范围中较低的界，被编码为一个无符号整数，在第三个到第八个比特中表示（见 C.22.3.1）。此无符号整数值为 47_{10}，而长度为 48_{10}（低界为 1）。 编码后的 UTF-8 字符（URI 的字符）的 48_{10} 个八位组所在的位置是从第 08_{16} 个八位组到第 37_{16} 个八位组。	
38	01111000 (78)	此八位组是对 **document** 信息项的一个孩子的初始编码。 所处位置为 38_{16} 的八位组，其值为 78_{16}，其中第一个比特为一个'0'（标识），表示有 **document** 信息项的一个孩子存在，且该孩子是一个元素（**element**）信息项（见 C.2.11.2）。第二个比特为'1'，表示此元素信息项具有属性（见 C.3.3）。第三个到第六个比特为'1110'，并在第七个和第八个比特跟随两个'00'（填充），表示命名空间 **attribute** 信息项存在（见 C.3.4.1）。	**element** 信息项，且具有 **[namespace attribute]** 属性

表 D.5－编码详述

	八 位 组	描 述	XML信息集或XML
39 3a 3b	11001111 (cf) 10000001 (81) 10000001 (81)	这些八位组是对命名空间属性信息项的编码，且具有被索引的 **[prefix]** 和 **[normalized value]** 属性。 所处位置为 39_{16} 的八位组，其值为 cf_{16}，其中第一个到第六个比特为'110011'（标识），表示一个命名空间 **attribute** 信息项存在（见 C.3.4.2）。第七个比特为'1'，表示 **[prefix]** 属性存在。第八个比特为'1'，表示**[normalized value]** 属性存在。 所处位置为 $3a_{16}$ 的八位组，其值为 81_{16}，其中第一个比特为'1'，表示一个索引被编码，且放入 PREFIX 表的该索引将标识**[prefix]** 属性（见 C.13.4）。第二个比特为'0'，表示该索引大于或等于 1_{10} ，且小于或等于 64_{10}，同时该索引被编码为一个无符号整数，放在第三个到第八个比特中（见 C.25.2）。该无符号整数为 1_{10}，而该索引为 2_{10}（低界为 1_{10}），这就会导致在从 PREFIX 表中解引用时，可以得到 **[prefix]** 属性为"res"。 所处位置为 $3b_{16}$ 的八位组，其值为 81_{16}，其中第一个比特为'1'，表示有一个索引被编码，且放入 NAMESPACE NAME 表的该索引将标识**[normalized value]** 属性（见 C.13.4）。 第二个比特为'0'，表示该索引大于或等于 1_{10} ，且小于或等于 64_{10}，同时该索引被编码为一个无符号整数，在第三个到第八个比特中表示（见 C.25.2）。该无符号整数为 1_{10} ，而该索引为 2_{10} （低界为 1_{10}），这就会导致在从 NAMESPACE NAME 表中解引用时，可以得到**[normalized value]** 属性为" /…ResponseCode:1.0"。	**xmlns:res= "….ResponseCode:1:0"**
3c 3d 3e	11001111 (cf) 10000010 (82) 10000010 (82)	这些八位组是对命名空间 **attribute** 信息项的编码，该信息项具有被索引的**[prefix]** 和 **[normalized value]** 属性。 **[prefix]** 属性的索引为 3_{10}，这就会导致在从 PREFIX 表中解引用时，可以得到一个值"cbc"。 **[normalized value]** 属性的索引为 3_{10}，这就会导致在从 NAMESPACE NAME 表中解引用时，可以得到一个值"….sicComponents:1.0"。	**xmlns:cbc= "….sicComponents:1:0"**
3f 40 41	11001111 (cf) 10000011 (83) 10000011 (83)	这些八位组是对命名空间 **attribute** 信息项的编码，且具有被索引的 **[prefix]** 和 **[normalized value]** 属性。	**xmlns:cac= "….ateComponents:1:0"**
42 43 44	11001111 (cf) 10000100 (84) 10000100 (84)	这些八位组是对命名空间 **attribute** 信息项的编码，且具有被索引的 **[prefix]** 和 **[normalized value]** 属性。	**xmlns:cur= "….CurrencyCode:1:0"**
45 46 47	11001111 (cf) 10000101 (85) 10000101 (85)	这些八位组是对命名空间 **attribute** 信息项的编码，且具有被索引的 **[prefix]** 和 **[normalized value]** 属性。	**xmlns:xsi= "….Schema-instance"**
48 49	11001101 (cd) 10000110 (86)	这些八位组是对命名空间 **attribute** 信息项的编码，且具有被索引的 **[prefix]** 和 **[normalized value]** 属性。 所处位置为 48_{16} 的八位组，其值为 cd_{16}，其中第七个比特为'0'，表示**[prefix]** 属性缺失，且第八个比特为'1'，表示**[normalized value]** 属性存在。	**xmlns="….Order:1:0"**
4a	11110000 (f0)	此八位组是对命名空间 **attribute** 信息项序列的终止符的编码。 所处位置为 $4a_{16}$ 的八位组，其值为 $f0_{16}$，其中前四个比特（第一个到第四个比特）为'1111'（终止符），是表示该序列的终止。六个'0'（填充）中的前四个出现在第五个到第八个比特中（见 C.3.4.3）。	

表 D.5－编码详述

	八 位 组	描 述	XML信息集或XML
4b	00000000 (00)	此八位组是对 **element** 信息项中的一个被索引的限定名字的编码。 所处位置为 $4b_{16}$ 的八位组，其值为 00_{16}，其中第一个和第二个比特是六个'0'（填充）中的最后两个（见 C.3.4.3）。第三个比特为'0'，表示此限定名字不是一个按照字面含义的限定名字（见 C.18.3）而是被索引的。该索引大于或等于 1_{10} ，且小于或等于 32_{10}，同时该索引被编码为一个无符号整数，放在第四个到第八个比特中（见 C.27.2）。此无符号整数为 0_{10}，而此索引为 1_{10}（低界为 1_{10}），这就会导致在从 ELEMENT NAME 表中解引用时，可以得到**[namespace name]** 属性为"….Order:1.0"，且**[local name]** 属性为"Order"的一个限定名字（对于此限定名字，没有**[prefix]** 属性）。	**<Order ….**
4c 4d 4e 4f 92	00000000 (00) 00001000 (08) 01111011 (3b) 01110101 (75) 01101000 (64)	这些八位组是对一个 **attribute** 信息项的编码，且具有被索引的限定名字以及一个**[normalized value]** 属性。**attribute** 信息项的存在在位置为 38_{16} 的八位组中表示（第二个比特为'1'）。 所处位置为 $4c_{16}$ 的八位组，其值为 00_{16}，其中第一个比特为'0'（标识），表示存在一个 **attribute** 信息项（见 C.3.6.1）。第二个比特为'0'，表示此限定名字不是一个按照字面含义的限定名字（见 C.17.3）而是被索引的。该索引大于或等于 1_{10} ，且小于或等于 64_{10}，同时该索引被编码为一个无符号整数，在第三个到第八个比特中表示（见 C.25.2）。此无符号整数为 0_{10} ，而此索引为 1_{10} （低界为 1_{10}），这就会导致在从 ATTRIBUTE NAME 表中解引用时，可以得到**[prefix]** 属性为"xsi"，**[namespace name]** 属性为"….Schema-instance"，且**[local name]** 属性为"schemaLocation"的一个限定名字。 所处位置为 $4d_{16}$ 的八位组，其值为 08_{16}，是对**[normalized value]** 属性的一个未标识串或索引（见 C.14）的初始编码。其中第一个比特为'0'，表示存在一个按照字面含义的限定名字（见 C.14.3）。第二个比特为'0'，表示该按照字面含义的字符串不应被加入到 ATTRIBUTE VALUE 表中。第三个和第四个比特都是'0'，表示串的编码格式为 UTF-8（见 C.19.3.1）。第五个和第六个比特为'1' 和'0'，分别表示被编码的 UTF-8 字符（**[normalized value]** 属性）的八位组长度大于或等于 9_{10} 个八位组，且小于或等于 264_{10} 个八位组，且该长度在减去较低的界后，被编码为一个无符号整数，放在下一个八位组的八个比特中（见 C.23.3.2）。第七个到第八个比特为'0'（填充）（见 C.23.3.2）。 所处位置为 $4e_{16}$ 的八位组，其值为 $3b_{16}$，是对无符号整数的编码。编码后的 UTF-8 字符的八位组的长度为 68_{10} （低界为 9_{10}）。 编码后的 UTF-8 字符（为**[normalized value]**属性）的 68_{10} 个八位组所在的位置是从第 $4f_{16}$ 个八位组到第 92_{16} 个八位组。	**xsi:schemaLocation="….**
93	11110000 (f0)	此八位组是对 **attribute** 信息项序列的终止符的编码。 所处位置为 93_{16} 的八位组，其值为 $f0_{16}$，其中前四个比特（第一个到第四个比特）为'1111'，是表示该序列的终止符。由于 **Order attribute** 信息项具有孩子，因此出现四个'0'（填充）（在第五个到第八个比特中）（见 D.3.2）。	

D.4.3.2 BuyerParty元素信息项中的Address元素信息项的编码

下面的解释详细说明了快速信息集文档中，**BuyerParty** 元素信息项中的 **Address element** 信息项的编码。尤其是对元素（**element**）信息项和字符（**character**）信息项的编码进行了解释。表 D.6 给出了对 D.3.2 中 **BuyerParty element** 信息项中的 **Address element** 信息项进行编码的快速信息集文档的片段。表 D.7 详细解释了此编码。以 XML 1.0 表示的片段如下所示：

```
<cac:Address>
        <cbc:StreetName>Marsh Lane</cbc:StreetName>
        <cbc:CityName>Nowhere</cbc:CityName>
        <cbc:PostalZone>NR18 4XX</cbc:PostalZone>
        <cbc:CountrySubentity>Norfolk</cbc:CountrySubentity>
</cac:Address>
```

表 D.6－八位组片段（16进制字符）

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
0000c0	070882074d61727368204c616e65f00982044e6f77686572
0000e0	65f00a82054e52313820345858f00b82044e6f72666f6c6bff

表 D.7－编码详述

	八 位 组	描 述	XML信息集或XML
c8	00000111 (07)	此八位组是对 **Address element** 信息项的编码。 所处位置为 $c8_{16}$ 的八位组，其值为 07_{16}，其中第一个比特为'0'（标识）表示有一个元素信息项的孩子（即 **Party element** 信息项的孩子），并且此孩子也是一个元素信息项（见 C.3.7.2）。第二个比特为'0'，表示此 **element** 信息项没有属性（见 C.3.3）。第三个比特为'0'，表示此限定名字不是一个按照字面含义的限定名字（见 C.18.3），而是被索引的。该索引大于或等于 1_{10}，且小于或等于 32_{10}，同时该索引被编码为一个无符号整数，放在第四个到第八个比特中（见 C.27.2）。此无符号整数为 7_{10}，而此索引为 8_{10}（低界为 1_{10}），这就会导致在从 ELEMENT NAME 表中解引用时，可以得到**[prefix]** 属性为 "csc"，一个 **[namespace name]** 属性为 "….gateComponents:1.0"，且一个**[local name]** 属性为" Address"的一个限定名字。	**<cac:Address>**
c9	00001000 (08)	此八位组是对 **StreetName element** 信息项的编码。 此 **element** 信息项具有索引值为 9_{10}，这就会导致在从 ELEMENT NAME 表中解引用时，可以得到**[prefix]** 属性为"csc"，一个 **[namespace name]** 属性为" ….BasicComponents:1:0"，且一个**[local name]** 属性为" StreetName"的一个限定名字。	**<cbc:StreetName>**

表 D.7－编码详述

	八 位 组	描 述	XML信息集或XML
ca cb cc d5	10000010 (82) 00000111 (07) 01001101 (4d) 01100101 (65)	这些八位组是对 **StreetName** 元素信息项中的 **character** 信息项的编码。 所处位置为 ca_{16} 的八位组，其值为 82_{16}，其中前两个比特（第一个到第二个比特）为'10'（标识），表示有一个元素信息项的孩子（即 **StreetName** 元素信息项的孩子），且此孩子为一个 **character** 信息项的块（见 C.3.7.5）。第三个比特为'0'，表示提供了一个按照字面含义的字符串（见 C.15.3）。第四个比特为'0'，表示此按照字面含义的字符串不能被加入到 CONTENT CHARACTER CHUNK 表中。第五个和第六个比特都是'0'，表示该块的编码格式为 UTF-8（见 C.20.3.1）。第七个和第八个比特为'1'和'0'，分别表示被编码的 UTF-8 字符（**character** 信息项的块）的八位组的长度大于或等于 3_{10} 个八位组且小于或等于 258_{10} 个八位组，同时该长度在减去较低的界后，被编码为一个无符号整数（见 C.24.3.2），且放在下一个八位组的八个比特中。 所处位置为 cb_{16} 的八位组，其值为 07_{16}，是一个无符号整数。已经编码的 UTF-8 字符的八位组长度为 10_{10} （低界为 3_{10}）。 编码后的 UTF-8 字符的 10_{10} 个八位组所在的位置是从第 cc_{16} 个八位组到第 $d5_{10}$ 个八位组。	**character** 信息项 "Marsh Lane"
d6	11110000 (f0)	此八位组是 **StreetName element** 信息项的终止符。 所处位置为 $d6_{16}$ 的八位组，其值为 $f0_{16}$，其中前四个比特（第一个到第四个比特）为'1111'（终止符），是 **StreetName element** 信息项的终止符（见 C.3.8）。第五个到第八个比特为'0'（填充），因为还存在另外一个（对等的）孩子（即 **CityName** 元素信息项）（见 C.3.7.1）。	**</cbc:StreetName>**
d7	00001001 (09)	此八位组是对 **CityName element** 信息项的编码。 此元素信息项具有索引值为 10_{10}，这就会导致在从 ELEMENT NAME 表中解引用时，可以得到 **[prefix]** 属性为"cbc"，一个 **[namespace name]** 属性为" ….BasicComponents:1:0"，且一个 **[local name]** 属性为" CityName"的一个限定名字。	**<cbc:CityName>**
d8 d9 da e0	10000010 (82) 00000100 (04) 01001110 (4e) 01100101 (65)	这些八位组是对 **CityName element** 信息项中 **character** 信息项的编码。 编码后的 UTF-8 字符的 7_{10} 个八位组所在的位置是从第 da_{16} 个八位组到第 $e0_{16}$ 个八位组。	**character** 信息项 "Nowhere"
e1	11110000 (f0)	此八位组是 **CityName element** 信息项的终止符。	**</cbc:CityName>**
e2	00001010 (0a)	此八位组是对 **PostalZone element** 信息项的编码。 此 **element** 信息项具有索引值为 11_{10}，这就会导致在从 ELEMENT NAME 表中解引用时，可以得到 **[prefix]** 属性为"cbc"，一个 **[namespace name]** 属性为" ….BasicComponents:1:0"，且一个 **[local name]** 属性为" PostalZone "的一个限定名字。	**<cbc:PostalZone>**
e3 e4 e5 .... ec	10000010 (82) 00000101 (05) 01001110 (4e) 01011000 (58)	这些八位组是对 **PostalZone element** 信息项中 **character** 信息项的编码。 编码后的 UTF-8 字符的 8_{10} 个八位组所在的位置是从第 $e5_{16}$ 个八位组到第 ec_{16} 个八位组。	**character** 信息项 "NR18 4XX"
ed	11110000 (f0)	此八位组是 **PostalZone element** 信息项的终止符。	**</cbc:PostalZone>**

表 D.7－编码详述

	八 位 组	描 述	XML信息集或XML
ee	00001011 (0b)	此八位组是对 **CountrySubentity element** 信息项的编码。 此 **element** 信息项具有索引值为 12_{10}，这就会导致在从 ELEMENT NAME 表中解引用时，可以得到 **[prefix]** 属性为"cbc"，一个 **[namespace name]** 属性为" ….BasicComponents:1:0"，且一个 **[local name]** 属性为" CountrySubentity"的一个限定名字。	**<cbc:CountrySubentity>**
ef f0 f1 .... f7	10000010 (82) 00000100 (04) 01001110 (4e) 01101011 (6b)	这些八位组是对 **CountrySubentity element** 信息项中 **character** 信息项的编码。 编码后的 UTF-8 字符的 7_{10} 个八位组所在的位置是从第 $f1_{16}$ 个八位组到第 $f7_{16}$ 个八位组。	**Character** 信息项 "Norfolk"
f8	11111111 (ff)	此八位组是 **CountrySubentity element** 信息项以及 **Address element** 信息项的终止符。 所处位置为 $f8_{16}$ 的八位组，其值为 ff_{16}，其中前四个比特（第一个到第四个比特）为'1111'（终止符），是 **CountrySubentity element** 信息项的终止符（见 C.3.8）。后四个比特（从第五个到第八个比特）为'1111'，是 **Address element** 信息项的终止符（见 C.3.8）。	**</cbc:CountrySubentity>** **</cac:Address>**

D.5 不具备初始词汇的UBL定单快速信息集文档

快速信息集文档的八位组（16 进制字符）在 D.5.1 中给出。关于 D.5.1 中某些八位组序列的详细解释在 D.5.2 中给出。此快速信息集文档的最终词汇与之前的快速信息集文档的最终词汇应当是相同的，因为这些表在外部词汇中的词汇表索引是按照相同的顺序产生的。由于会有串嵌入到快速信息集文档中，这将会导致较长的长度。包含串的代价为 635_{10} 个字节（此快速信息集文档的长度减去之前的快速信息集文档的长度），大致占到了文档长度的一半（对于更大的文档而言，这种差异会小一些，因为词汇表将趋于一个固定的代价）。与之前的快速信息集文档不同的是，本文档可以被认为是自我描述的，因为不需要任何外部信息（外部词汇）就可以产生 XML 信息集。

D.5.1 快速信息集文档的八位组（16进制字符）

表 D.8 给出了 D.3 中给出的 UBL 定单示例的快速信息集文档的八位组。

注 — 包含了对应于信息项的标识符和终止符比特的那些16进制字符用下划线标识。

表 D.8－快速信息集文档的八位组（16进制字符）

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000000	e001000000078cf027265733e75726e3a6f617369733a6e616d65733a74633a75
000020	626c3a636f64656c6973743a41636b6e6f776c656467656d656e74526573706f
000040	6e7365436f64653a313a30cf026362632f75726e3a6f617369733a6e616d6573
000060	3a74633a75626c3a436f6d6d6f6e4261736963436f6d706f6e656e74733a313a
000080	30cf026361633375726e3a6f617369733a6e616d65733a74633a75626c3a436f
0000a0	6d6d6f6e41676772656761746543 6f6d706f6e656e74733a313a30cf02637572
0000c0	2f75726e3a6f617369733a6e616d65733a74633a75626c3a636f64656c697374
0000e0	3a43757272656e6379436f64653a313a30cf027873692868747 4703a2f2f7777
000100	772e77332e6f72672f323030312f584d4c536368656d612d696e7374616e6365
000120	cd1f75726e3a6f617369733a6e616d65733a74633a75626c3a4f726465723a31
000140	3a30f03d86044f726465727b85850d736368656d614c6f636174696f6e083b75
000160	726e3a6f617369733a6e616d65733a74633a75626c3a4f726465723a313a3020
000180	2e2e2f2e2e2f7873642f6d61696e646f632f55424c2d4f726465722d312e302e
0001a0	787364f03d860742757965727349448207533033 2d303334323537f03f828208
0001c0	4973737565446174658207323030332d30322d3033f03f838309427579657250
0001e0	617274793f838304506172747 93f8383085061727 4794e616d653f8282034e61
000200	6d65820e4a65727279204275696c64657220706c63ff3f838306416464726573
000220	733f828209537472656574e616d6582074d61727368204c616e65f03f828207
000240	436974794e616d6582044e6f7768657265f03f828209506f7374616c5a6f6e65
000260	82054e52313820345858f03f82820f436f756e747279537562656e7469747982
000280	044e6f72666f6c6bff3f838306436f6e7461637406820645766120427269636b
0002a0	ffff3f83830a53656c6c65725061727479040506821353706563696 16c697374
0002c0	2057696e646f777320706c63ff073f82820b4275696c64696e674e616d65820b
0002e0	536e6f7768696c6c20576f726b73f009820b4c6974746c6520536e6f72696e67
000300	f00a8204534d3220334e57f00b8207576865726573686972656ffff3f83830744
000320	656c69766572793f828218526571756573746564465 6c697665727944617465
000340	54696d65821032303033 2d30322d32345430303a30303a3030f03f83830e4465
000360	6c697665727941646472657373 08820a52697665727369646520526 42ef00e82
000380	17506c6f742031372c20576869746577617465722045737461746 5f009820657
0003a0	68657473746f6e65f00b82064d6964646c65736578fff03f8383084f72646572
0003c0	4c696e653f8383074c696e654974656d3f838382 9041f07f8282075175616e74
0003e0	697479780f7175616e74697479556e6974436f646543756e6974f09032f03f83
000400	83034974656d3f83831853656c6c6572734974656d4964656e74696669636174
000420	696f6e3f838301494492023233365756f03f838310506879736963616c417474
000440	7269627574653f83830a417474726962757465494492 01776f6f64f03f82820a
000460	446573637269707469 6f6e92 01736f6674ff191a820366696e697368f01b8203
000480	7072696d6564ff191a8205666974746 96e6773f01b920273617469 6eff191a82
0004a0	04676c617a696e67f01b820373696e676c65ffffff1213149042f0550180f090
0004c0	33f016171892023334305457f0191a920168616e64f01b915248ff191aa3f01b
0004e0	920168617264ff191a820366696e697368f01b920273746 1696eff191a820566
000500	697474696e6773f01b92026272617373ff191a8204676c617a696e67f01b8203
000520	646f75626c65ffffffff
00052a	

D.5.2 编码的解释

D.5.2.1 document 信息项和Order元素信息项的编码

下列解释详细说明了快速信息集文档和根元素信息项的初始编码。特别的，对一个 **document** 信息项，一系列命名空间（**namespace**）信息项，一个元素（**element**）信息项，以及一个属性（**attribute**）信息项的编码进行了解释。表 D.9 给出了对 D.3.2 中 **document** 信息项和 **Order** 元素信息项进行编码的快速信息集文档的片段。表 D.10 详细解释了此编码。以 XML 1.0 表示的片段如下所示：

```
<Order xmlns:res="urn:oasis:names:tc:ubl:codelist:AcknowledgementResponseCode:1:0"
xmlns:cbc="urn:oasis:names:tc:ubl:CommonBasicComponents:1:0"
xmlns:cac="urn:oasis:names:tc:ubl:CommonAggregateComponents:1:0"
xmlns:cur="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oasis:names:tc:ubl:Order:1:0"
xsi:schemaLocation="urn:oasis:names:tc:ubl:Order:1:0 ../../xsd/maindoc/UBL-Order-1.0.xsd">
```

表 D.9－八位组片段（16进制字符）

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000000	e001000000078cf027265733e75726e3a6f617369733a6e616d65733a74633a75
000020	626c3a636f64656c6973743a41636b6e6f776c656467656d656e74526573706f
000040	6e7365436f64653a313a30cf026362632f75726e3a6f617369733a6e616d6573
000060	3a74633a75626c3a436f6d6d6f6e4261736963436f6d706f6e656e74733a313a
000080	30cf026361633375726e3a6f617369733a6e616d65733a74633a75626c3a436f
0000a0	6d6d6f6e41676772656761746543 6f6d706f6e656e74733a313a30cf02637572
0000c0	2f75726e3a6f617369733a6e616d65733a74633a75626c3a636f64656c697374
0000e0	3a43757272656e6379436f64653a313a30cf027873692868747 4703a2f2f7777
000100	772e77332e6f72672f323030312f584d4c536368656d612d696e7374616e6365
000120	cd1f75726e3a6f617369733a6e616d65733a74633a75626c3a4f726465723a31
000140	3a30f03d86044f726465727b85850d736368656d614c6f636174696f6e083b75
000160	726e3a6f617369733a6e616d65733a74633a75626c3a4f726465723a313a3020
000180	2e2e2f2e2e2f7873642f6d61696e646f632f55424c2d4f726465722d312e302e
0001a0	787364f0

表 D.10－编码详述

	八 位 组	描 述	XML信息集或XML
00 01	11100000 (e0) 00000000 (00)	这些八位组应出现在每个快速信息集文档的起始位置（见 12.6）。	**document** 信息项
02 03	00000000 (00) 00000001 (01)	这些八位组为版本号的编码（见 12.9）。	
04	00000000 (00)	此八位组是对存在一个初始词汇的编码，以及对 **Document** 类型的其他组件的编码。 所处位置为 04_{16} 的八位组，其值为 00_{16}，其中第一个比特为一个'0'（填充）（见 12.8 节）。第二个比特到第八个比特为'0000000'，表示 **Document** 类型的所有可选组件都缺失（包括 **initial-vocabulary** 组件，其缺失表现在第三个比特上，见 C.2.3）。	
05	01111000 (78)	此八位组是对 **document** 信息项的一个孩子的初始编码。 所处位置为 05_{16} 的八位组，其值为 78_{16}，其中第一个比特为一个'0'（标识），表示有 **document** 信息项的一个孩子存在，且该孩子是一个元素（**element**） 信息项（见 C.2.11.2）。第二个比特为'1'，表示此元素信息项具有属性（见 C.3.3）。第三个到第六个比特为'1110'，并在第七个和第八个比特跟随两个'00'（填充），表示命名空间属性信息项存在（见 C.3.4.1）。	**element** 信息项，具有 **[namespace attribute]**属性。

表 D.10－编码详述

	八 位 组	描 述	XML信息集或XML
06 07 08 .... 0a 0b 0c 4a	11001111 (cf) 00000010 (02) 01110010 (72) .... 01110010 (73) 00111110 (3e) 01110101 (75) 01110000 (30)	这些八位组是对命名空间 **attribute** 信息项的编码，且具有字面含义的**[prefix]** 和 **[normalized value]** 属性。 所处位置为 06_{16} 的八位组，其值为 cf_{16}，其中第一个到第六个比特为'110011'（标识），表示一个命名空间属性信息项存在（见 C.3.4.2）。第七个比特为'1'，表示 **[prefix]** 属性存在。第八个比特为'1'，表示 **[normalized value]** 属性存在。 所处位置为 07_{16} 的八位组，其值为 02_{16}，其中第一个比特为'0'，表示对**[prefix]** 属性的一个具有字面含义的字符串进行编码（见 C.13.3）。第二个比特为'0'，表示被编码的 UTF-8 字符的长度大于或等于 1_{10}，且小于或等于 64_{10}，同时该长度被编码为一个无符号整数，放在第三个到第八个比特中（见 C.22.3.1）。该无符号整数为 2_{10}，而该长度为 3_{10}（低界为 1_{10}）。 编码后的 UTF-8 字符的这 3_{10} 个八位组（**[prefix]**属性）所在的位置是从第 08_{16} 个八位组到第 $0a_{16}$ 个八位组。串"res"将被加入到 PREFIX 表中（其索引为 2_{10}）。 所处位置为 $0b_{16}$ 的八位组，其值为 $3e_{16}$，其中第一个比特为'0'，表示对**[normalized value]** 属性的一个具有字面含义的字符串进行编码（见 C.13.3）。第二个比特为'0'，表示被编码的 UTF-8 字符的长度大于或等于 1_{10}，且小于或等于 64_{10}，同时该长度被编码为一个无符号整数，放在第三个到第八个比特中（见 C.22.3.1）。该无符号整数为 62_{10}，而该长度为 63_{10}（低界为 1_{10}）。 编码后的 UTF-8 字符的这 63_{10} 个八位组（**[normalized value]**属性）所在的位置是从第 $0c_{16}$ 个八位组到第 $4a_{16}$ 个八位组。串"….ResponseCode:1:0"将被加入到 NAMESPACE NAME 表中（其索引为 2_{10}）。	**xmlns:res=** **"….ResponseCode:1:0"**
4b 4c 4d 4f 50 51 80	11001111 (cf) 00000010 (02) 01100011 (63) .... 01100011 (63) 00101111 (2f) 01110101 (75) 01110000 (30)	这些八位组是对命名空间属性信息项的编码，该信息项具有字面含义的 **[prefix]** 和 **[normalized value]** 属性。 编码后的 UTF-8 字符的 3_{10} 个八位组（**[prefix]**属性）所在的位置是从第 $4c_{16}$ 个八位组到第 $4f_{16}$ 个八位组。串"cbc"将被加入到 PREFIX 表中（其索引为 3_{10}）。 编码后的 UTF-8 字符的 48_{10} 个八位组（**[normalized value]**属性）所在的位置是从第 51_{16} 个八位组到第 80_{16} 个八位组。串"….sicComponents:1:0"将被加入到 NAMESPACE NAME 表中（其索引为 3_{10}）。	**xmlns:cbc=** **"….sicComponents:1:0"**
81 82 83 .... 85 86 87 .... ba	11001111 (cf) 00000010 (02) 01100011 (63) .... 01100011 (63) 00110011 (33) 01110101 (75) 01110000 (30)	这些八位组是对命名空间属性信息项的编码，该信息项具有字面含义的 **[prefix]** 和 **[normalized value]** 属性。 编码后的 UTF-8 字符的 3_{10} 个八位组（**[prefix]**属性）所在的位置是从第 83_{16} 个八位组到第 85_{16} 个八位组。串"cac"将被加入到 PREFIX 表中（其索引为 4_{10}）。 编码后的 UTF-8 字符的 52_{10} 个八位组（**[normalized value]**属性）所在的位置是从第 87_{16} 个八位组到第 ba_{16} 个八位组。串"….ateComponents:1:0"将被加入到 NAMESPACE NAME 表中（其索引为 4_{10}）。	**xmlns:cac=** **"….ateComponents:1:0"**
bb bc bd .... bf c0 c1 .... f0	11001111 (cf) 00000010 (02) 01100011 (63) .... 01100011 (63) 00101111 (2f) 01110101 (75) 01110000 (30)	这些八位组是对命名空间属性信息项的编码，该信息项具有字面含义的 **[prefix]** 和 **[normalized value]** 属性。 编码后的 UTF-8 字符的 3_{10} 个八位组（**[prefix]**属性）所在的位置是从第 bd_{16} 个八位组到第 bf_{16} 个八位组。串"cur"将被加入到 PREFIX 表中（其索引为 5_{10}）。 编码后的 UTF-8 字符的 48_{10} 个八位组（**[normalized value]**属性）所在的位置是从第 $c1_{16}$ 个八位组到第 $f0_{16}$ 个八位组。串"….CurrencyCode:1:0"将被加入到 NAMESPACE NAME 表中（其索引为 5_{10}）。	**xmlns:cur=** **"….CurrencyCode:1:0"**

表 D.10－编码详述

	八 位 组	描 述	XML信息集或XML
f1 f2 f3 .... f5 f6 f7 .... 11f	11001111 (cf) 00000010 (02) 01111000 (78) .... 01101001 (69) 00101000 (28) 01101000 (68) 01110111 (77)	这些八位组是对命名空间属性信息项的编码，该信息项具有字面含义的 **[prefix]** 和 **[normalized value]** 属性。 编码后的 UTF-8 字符的 3_{10} 个八位组（**[prefix]**属性）所在的位置是从第 $f3_{16}$ 个八位组到第 $f5_{16}$ 个八位组。串"xsi"将被加入到 PREFIX 表中（其索引为 6_{10}）。 编码后的 UTF-8 字符的 41_{10} 个八位组（**[normalized value]**属性）所在的位置是从第 $f7_{16}$ 个八位组到第 $11f_{16}$ 个八位组。串"….Schema-instance"将被加入到 NAMESPACE NAME 表中（其索引为 6_{10}）。	**xmlns:xsi="….Schema-instance"**
120 121 122 141	11001101 (cd) 00011111 (1f) 01110101 (75) 00110000 (30)	这些八位组是对命名空间属性信息项的编码，该信息项具有被索引的**[normalized value]** 属性。 编码后的 UTF-8 字符的 32_{10} 个八位组（**[normalized value]**属性）所在的位置是从第 122_{16} 个八位组到第 141_{16} 个八位组。串"….Order:1:0"将被加入到 NAMESPACE NAME 表中（其索引为 7_{10}）。	**xmlns="….Order:1:0"**
142	11110000 (f0)	此八位组是对命名空间属性信息项序列的终止符的编码。 所处位置为 142_{16} 的八位组，其值为 $f0_{16}$，其中前四个比特（第一个到第四个比特）为'1111'（终止符），是表示该序列的终止。六个'0'（填充）中的前四个出现在第五个到第八个比特中（见 C.3.4.3）。	
143 144 145 146 .... 14a	00111101 (3d) 10000110 (86) 00000100 (04) 01001111 (4f) 01110010 (72)	这些八位组是对元素信息项中的一个具有字面含义的限定名字的编码。 所处位置为 143_{16} 的八位组，其值为 $3d_{16}$，其中第一个和第二个比特是六个'0'（填充）中的最后两个（见 C.3.4.3）。第三个到第六个比特为'1111'，表示该限定名字为一个具有字面含义的限定名字（见 C.18.3）。第七个比特为'0'，表示此限定名字没有一个**[prefix]** 属性。第八个比特为'1'，表示此限定名字具有一个**[namespace name]** 属性。 所处位置为 144_{16} 的八位组，其值为 86_{16}，其中第一个比特为'1'，表示此**[namespace name]** 属性不是一个按照字面含义的串，而是被索引的（见 C.13.4）。第二个比特为'0'，表示该索引大于或等于 1_{10}，且小于或等于 64_{10}，同时该索引被编码为一个无符号整数，放在第三个到第八个比特中（见 C.25.2）。此无符号整数为 6_{10}，而此索引为 7_{10}（低界为 1_{10}），这就会导致在从 NAMESPACE NAME 表中解引用时，可以得到**[namespace name]** 属性为"….Order:1.0"。 所处位置为 145_{16} 的八位组，其值为 04_{16}，其中第一个比特为'0'，表示对一个具有字面含义的**[local name]** 属性进行编码（见 C.13.3）。第二个比特为'0'，表示被编码的 UTF-8 字符的长度大于或等于 1_{10}，且小于或等于 64_{10}，同时该长度被编码为一个无符号整数，放在第三个到第八个比特中（见 C.22.3.1）。此无符号整数为 4_{10}，而此长度为 5_{10}（低界为 1_{10}）。 编码后的 UTF-8 字符的 5_{10} 个八位组（**[local name]**属性）所在的位置是从第 146_{16} 个八位组到第 $14a_{16}$ 个八位组。串" Order"将被加入到 LOCAL NAME 表中（其索引为 1_{10}）。 没有**[prefix]** 属性，有一个**[namespace name]** 属性为"….Order:1:0"（索引为 7_{10}），且有一个**[local name]** 属性为"Order"（索引为 1_{10}）的一个限定名字将被加入到 ELEMENT NAME 表中（索引为 1_{10}）。	**<Order ….**

表 D.10－编码详述

	八 位 组	描 述	XML信息集或XML
14b 14c 14d 14e 14f .... 15c 15d 15e 15f 1a2	01111011 (7b) 10000101 (85) 10000101 (85) 00001101 (0d) 01110011 (73) .... 01101110 (6e) 00001000 (08) 00111011 (3b) 01110101 (75) 01100100 (64)	这些八位组是对一个属性信息项的编码，该信息项具有一个按照字面含义的限定名字以及一个**[normalized value]** 属性。该属性信息项的存在由处于位置为 05_{16} （第二个比特为'1'）的八位组来表示。 所处位置为 $14b_{16}$ 的八位组，其值为 $7b_{16}$，其中第一个比特为'0'（标识），表示有一个属性信息项存在（见 C.3.6.1）。第二个到第五个比特为'1111'，表示该限定名字为一个具有字面含义的限定名字（见 C.17.3）。第六个比特为'0'（填充）（见 C.17.3）。第七个比特为'1'，表示此限定名字有一个**[prefix]** 属性。第八个比特为'1'，表示此限定名字有一个**[namespace name]** 属性。 所处位置为 $14c_{16}$ 的八位组，其值为 85_{16}，其中第一个比特为'1'，表示**[prefix]** 属性不是一个按照字面含义的串，而是被索引的（见 C.13.4）。第二个比特为'0'，表示该索引大于或等于 1_{10}，且小于或等于 64_{10}，同时该索引被编码为一个无符号整数，放在第三个到第八个比特中（见 C.25.2）。此无符号整数为 5_{10}，而此索引为 6_{10} （低界为 1_{10}），这就会导致在从 NAMESPACE NAME 表中解引用时，可以得到一个**[prefix]** 属性为"xsi"。 所处位置为 $14d_{16}$ 的八位组，其值为 85_{16}，其中第一个比特为'1'，表示**[namespace name]** 属性不是一个按照字面含义的串，而是被索引的（见 C.13.4）。第二个比特为'0'，表示该索引大于或等于 1_{10}，且小于或等于 64_{10}，同时该索引被编码为一个无符号整数，放在第三个到第八个比特中（见 C.25.2）。此无符号整数为 5_{10}，而此索引为 6_{10}（低界为 1_{10}），这就会导致在从 NAMESPACE NAME 表中解引用时，可以得到一个**[namespace name]** 属性为"….Schema-instance "。 所处位置为 $14e_{16}$ 的八位组，其值为 $0d_{16}$，其中第一个比特为'0'，表示**[local name]** 属性是一个按照字面含义的字符串，并对其进行编码（见 C.13.3）。第二个比特为'0'，表示已经编码的 UTF-8 字符的长度大于或等于 1_{10}，且小于或等于 64_{10}，同时该长度被编码为一个无符号整数，放在第三个到第八个比特中（见 C.22.3.1）。此无符号整数为 13_{10}，而此长度为 14_{10} （低界为 1_{10}）。 编码后的 UTF-8 字符的 14_{10} 个八位组（**[local name]**属性）所在的位置是从第 $14f_{10}$ 个八位组到第 $15c_{10}$ 个八位组。串" SchemaLocation "将被加入到 LOCAL NAME 表中（其索引为 2_{10}）。 具有一个**[prefix]** 属性为"xsi" （其索引为 6_{10}），一个**[namespace name]** 属性为"….Schema-instance"（索引为 6_{10}），以及一个**[local name]** 属性为" schemaLocation"（索引为 2_{10}）的一个限定名字将被加入到 ATTRIBUTE NAME 表中（索引为 1_{10}）。 所处位置为 $15d_{16}$ 的八位组，其值为 08_{16}，是对**[normalized value]** 属性的一个未标识串或索引（见 C.14）的初始编码。其中第一个比特为'0'，表示存在一个按照字面含义的限定名字（见 C.14.3）。第二个比特为'0'，表示该按照字面含义的字符串不应被加入到 ATTRIBUTE VALUE 表中。第三个和第四个比特都是'0'，表示串的编码格式为 UTF-8（见 C.19.3.1）。第五个和第六个比特为'1' 和'0'，分别表示被编码的 UTF-8 字符（**[normalized value]** 属性）的八位组长度大于或等于 9_{10} 个八位组，且小于或等于 264_{10} 个八位组，且该长度在减去较低的界后，被编码为一个无符号整数，放在下一个八位组的八个比特中（见 C.23.3.2）。第七个到第八个比特为'0'（填充）（见 C.23.3.2）。 所处位置为 $15e_{16}$ 的八位组，其值为 $3b_{16}$，是对该无符号整数的编码。编码后的 UTF-8 字符的八位组的长度为 68_{10} （低界为 9_{10}）。 编码后的 UTF-8 字符（为**[normalized value]**属性）的 68_{10} 个八位组所在的位置是从第 $15f_{16}$ 个八位组到第 $1a2_{16}$ 个八位组。	**xsi:schemaLocation="…."**

表 D.10－编码详述

	八 位 组	描 述	XML信息集或XML
1a3	11110000 (f0)	此八位组是对属性信息项序列的终止符的编码。 所处位置为 $1a3_{16}$ 的八位组，其值为 $f0_{16}$，其中前四个比特（第一个到第四个比特）为'1111'，是表示该序列的终止符。由于 **Order** 元素信息项具有孩子，因此出现四个'0'（填充）（在第五个到第八个比特中）（见 D.3.2）。	

D.5.2.2 BuyerParty元素信息项中的Address元素信息项的编码

下面的解释详细说明了快速信息集文档中，**BuyerParty** 元素信息项中的 **Address** 元素信息项的编码。尤其是对元素（**element**）信息项和字符（**character**）信息项的编码进行了解释。表 D.11 给出了对 D.3.2 中 **BuyerParty** 元素信息项中的 **Address** 元素信息项进行编码的快速信息集文档的片段。表 D.12 详细解释了此编码。以 XML 1.0 表示的片段如下所示：

```
<cac:Address>
        <cbc:StreetName>Marsh Lane</cbc:StreetName>
        <cbc:CityName>Nowhere</cbc:CityName>
        <cbc:PostalZone>NR18 4XX</cbc:PostalZone>
        <cbc:CountrySubentity>Norfolk</cbc:CountrySubentity>
</cac:Address>
```

表 D.11－八位组片段（16进制字符）

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
000200	3f838306416464726573
000220	733f828209537472656574e616d6582074d61727368204c616e65f03f828207
000240	436974794e616d6582044e6f7768657265f03f828209506f7374616c5a6f6e65
000260	82054e52313820345858f03f82820f436f756e747279537562656e7469747982
000280	044e6f72666f6c6bff

表 D.12－编码详述

	八 位 组	描 述	XML信息集或XML
216 217 218 219 21a 220	00111111 (3f) 10000011 (83) 10000011 (83) 00000110 (06) 01000001 (41) 01110011 (73)	这些八位组是对 **Address** 元素信息项的编码。 所处位置为 216_{16} 的八位组，其值为 $3f_{16}$，其中第一个比特为'0'（标识）表示有一个元素信息项的孩子（即 **Party** 元素信息项的孩子），并且此孩子也是一个元素信息项（见 C.3.7.2）。第二个比特为'0'，表示此元素信息项没有属性（见 C.3.3）。第三个到第六个比特为'1111'，表示此限定名字是一个按照字面含义的限定名字（见 C.18.3）。第七个比特为'1'，表示此限定名字有一个**[prefix]** 属性。第八个比特为'1'，表示此限定名字有一个**[namespace name]** 属性。 所处位置为 217_{16} 的八位组，其值为 83_{16}，其中第一个比特为'1'，表示**[prefix]** 属性不是一个按照字面含义的串，而是被索引的（见 C.13.4）。第二个比特为'0'，表示该索引大于或等于 1_{10}，且小于或等于 64_{10}，同时该索引被编码为一个无符号整数，放在第三个到第八个比特中（见 C.25.2）。此无符号整数为 3_{10}，而此索引为 4_{10}（低界为 1_{10}），这就会导致在从 PREFIX 表中解引用时，可以得到一个**[prefix]** 属性为"cac"。 所处位置为 218_{16} 的八位组，其值为 83_{16}，其中第一个比特为'1'，表示**[namespace name]** 属性不是一个按照字面含义的串，而是被索引的（见 C.13.4）。第二个比特为'0'，表示该索引大于或等于 1_{10}，且小于或等于 64_{10}，同时该索引被编码为一个无符号整数，放在第三个到第八个比特中（见 C.25.2）。此无符号整数为 3_{10}，而此索引为 4_{10}（低界为 1_{10}），这就会导致在从 NAMESPACE NAME 表中解引用时，可以得到一个**[namespace name]** 属性为"….ateComponents:1:0"。 所处位置为 219_{16} 的八位组，其值为 06_{16}，其中第一个比特为'0'，表示**[local name]** 属性是一个按照字面含义的字符串，并对其进行编码（见 C.13.3）。第二个比特为'0'，表示已经编码后的 UTF-8 字符的长度大于或等于 1_{10}，且小于或等于 64_{10}，同时该长度被编码为一个无符号整数，放在第三个到第八个比特中（见 C.22.3.1）。此无符号整数为 6_{10}，而此长度为 7_{10}（低界为 1_{10}）。 编码后的 UTF-8 字符（**[local name]**属性）的 7_{10} 个八位组所在的位置是从第 $21a_{16}$ 个八位组到第 220_{16} 个八位组。串"Address"将被加入到 LOCAL NAME 表中（其索引为 9_{10}）。 具有一个**[prefix]** 属性为"cac"（其索引为 4_{10}），一个**[namespace name]** 属性为"……..ateComponents:1:0"（索引为 4_{10}），以及一个**[local name]** 属性为"Order"（索引为 1_{10}）的限定名字将被加入到 ELEMENT NAME 表中（其索引为 1_{10}）。	**<cac:Address>**
221 222 223 224 225 .... 22e	00111111 (3f) 10000010 (82) 10000010 (82) 00001001 (09) 01000001 (53) 01100101 (65)	这些八位组是对 **StreetName** 元素信息项的编码。 **[local name]** 属性"StreetName"将被加入到 LOCAL NAME 表中（其索引为 10_{10}）。 具有一个**[prefix]** 属性为"cbc"（其索引为 3_{10}），一个**[namespace name]** 属性为"……..BasicComponents:1:0"（索引为 3_{10}），以及一个**[local name]** 属性为" StreetName"（索引为 10_{10}）的限定名字将被加入到 ELEMENT NAME 表中（其索引为 9_{10}）。	**<cbc:StreetName>**

表 D.12－编码详述

	八 位 组	描 述	XML信息集或XML
22f 230 231 .... 23a	10000010 (82) 00000111 (07) 01001101 (4d) 01100101 (65)	这些八位组是对 **StreetName** 元素信息项中的 **character** 信息项的编码。 所处位置为 $22f_{16}$ 的八位组，其值为 82_{16}，其中前两个比特（第一个到第二个比特）为'10'（标识），表示有一个元素信息项的孩子（即 **StreetName** 元素信息项的孩子），且此孩子为一个 **character** 信息项的块（见 C.3.7.5）。第三个比特为'0'，表示提供了一个按照字面含义的字符串（见 C.15.3）。第四个比特为'0'，表示此按照字面含义的字符串不能被加入到 CONTENT CHARACTER CHUNK 表中。第五个和第六个比特都是'0'，表示该块的编码格式为 UTF-8（见 C.20.3.1）。第七个和第八个比特为'1'和'0'，分别表示被编码的 UTF-8 字符（**character** 信息项的块）的八位组的长度大于或等于 3_{10} 个八位组且小于或等于 258_{10} 个八位组，同时该长度在减去较低的界后，被编码为一个无符号整数（见 C.24.3.2），且放在下一个八位组的八个比特中。 所处位置为 230_{16} 的八位组，其值为 07_{16}，是一个无符号整数。已经编码的 UTF-8 字符的八位组长度为 10_{10} （低界为 3_{10}）。 编码后的 UTF-8 字符的 10_{10} 个八位组所在的位置是从第 231_{16} 个八位组到第 $23a_{16}$ 个八位组。	**character** 信息项"Marsh Lane"
23b	1111<u>0000</u> (f0)	此八位组是 **StreetName** 元素信息项的终止符。	**</cbc:StreetName>**
23c 23d 23e 23f 240 247	00111111 (3f) 10000010 (82) 10000010 (82) 00000111 (07) 01000011 (43) 01100101 (65)	这些八位组是对 **CityName** 元素信息项的编码。 **[local name]** 属性"CityName"将被加入到 LOCAL NAME 表中（其索引为 10_{10}）。 具有一个**[prefix]** 属性为"cbc"（其索引为 3_{10}），一个**[namespace name]** 属性为"……BasicComponents:1:0"（索引为 3_{10}），以及一个 **[local name]** 属性为"CityName"（索引为 11_{10}）的限定名字将被加入到 ELEMENT NAME 表中（其索引为 10_{10}）。	**<cbc:CityName>**
248 249 24a 250	10000010 (82) 00000100 (04) 01001110 (4e) 01100101 (65)	这些八位组是对信息项中的 **character** 信息项的编码。 编码后的 UTF-8 字符的 7_{10} 个八位组所在的位置是从第 $24a_{16}$ 个八位组到第 250_{16} 个八位组。	**character** 信息项"Nowhere"
251	1111<u>0000</u> (f0)	此八位组是 **CityName** 元素信息项的终止符。	**</cbc:CityName>**
252 253 254 255 256 .... 25f	00111111 (3f) 10000010 (82) 10000010 (82) 00001001 (09) 01000011 (50) 01100101 (65)	这些八位组是对 **PostalZone** 元素信息项的编码。 **[local name]** 属性" PostalZone"将被加入到 LOCAL NAME 表中（其索引为 12_{10}）。 具有一个**[prefix]** 属性为"cbc"（其索引为 3_{10}），一个**[namespace name]** 属性为"……BasicComponents:1:0"（索引为 3_{10}），以及一个 **[local name]** 属性为" PostalZone "（索引为 12_{10}）的限定名字将被加入到 ELEMENT NAME 表中（其索引为 11_{10}）。	**<cbc:PostalZone>**
260 261 262 269	10000010 (82) 00000101 (05) 01001110 (4e) 01011000 (58)	这些八位组是对 **PostalZone** 元素信息项中的 **character** 信息项的编码。 编码后的 UTF-8 字符的 8_{10} 个八位组所在的位置是从第 262_{16} 个八位组到第 269_{16} 个八位组。	**character** 信息项"NR18 4XX"
26a	1111<u>0000</u> (f0)	此八位组是 **PostalZone** 元素信息项的终止符。	**</cbc:PostalZone>**

表 D.12－编码详述

	八 位 组	描 述	XML信息集或XML
26b 26c 26d 26e 26f 27e	00111111 (3f) 10000010 (82) 10000010 (82) 00001111 (0f) 01000011 (43) 01111001 (79)	这些八位组是对 **CountrySubentity** 元素信息项的编码。 **[local name]** 属性" CountrySubentity"将被加入到 LOCAL NAME 表中（其索引为 13_{10}）。 具有一个**[prefix]** 属性为"cbc"（其索引为 3_{10}），一个**[namespace name]** 属性为"……..BasicComponents:1:0"（索引为 3_{10}），以及一个**[local name]** 属性为"CountrySubentity"（索引为 13_{10}）的限定名字将被加入到 ELEMENT NAME 表中（其索引为 12_{10}）。	**<cbc:CountrySubentity>**
27f 280 281 287	10000010 (82) 00000100 (04) 01001110 (4e) 01101011 (6b)	这些八位组是对 **CountrySubentity** 元素信息项中的 **character** 信息项的编码。 编码后的 UTF-8 字符的 7_{10} 个八位组所在的位置是从第 281_{16} 个八位组到第 287_{16} 个八位组。	**character** 信息项"Norfolk"
288	11111111 (ff)	此八位组是**CountrySubentity** 元素信息项以及**Address** 元素信息项的终止符。 所处位置为 288_{16} 的八位组，其值为 ff_{16}，其中前四个比特（第一个到第四个比特）为'1111'（终止符），是 **CountrySubentity** 元素信息项的终止符（见 C.3.8）。后四个比特（从第五个到第八个比特）为'1111'，是 **Address** 元素信息项的终止符（见 C.3.8）。	**</cbc:CountrySubentity>** **</cac:Address>**

D.5.2.3 第一个LineItem元素信息项中的BuyersID元素信息项的编码

下面的解释详细说明了快速信息集文档中，第一个 **LineItem** 元素信息项中的 **BuyersID** 元素信息项的编码。尤其是对其**[local name]**属性已经先于本信息项被索引的元素信息项进行了解释。表 D.13 给出了对 D.3.2 中第一个 **LineItem** 元素信息项中的 **BuyersID** 元素信息项进行编码的快速信息集文档的片段。表 D.14 详细解释了此编码。以 XML 1.0 表示的片段如下所示：

```
<cac:LineItem>
    <cac:BuyersID>A</cac:BuyersID>
```

表 D.13－八位组片段（16进制字符）

	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
0003c0	3f8383074c696e654974656d3f8383829041f0

表 D.14－编码详述

	八 位 组	描 述	XML信息集或XML
3c4 3c5 3c6 3c7 3c8 3cf	00111111 (3f) 10000011 (83) 10000011 (83) 00001111 (07) 01001010 (4c) 01101101 (6d)	这些八位组是对 **LineItem** 元素信息项的编码。 **[local name]** 属性"LineItem"将被加入到 LOCAL NAME 表中（其索引为 21_{10}）。 具有一个**[prefix]** 属性为"cac"（其索引为 4_{10}），一个**[namespace name]** 属性为"……..ateComponents:1:0"（索引为 4_{10}），以及一个**[local name]** 属性为" LineItem"（索引为 21_{10}）的限定名字将被加入到 ELEMENT NAME 表中（其索引为 20_{10}）。	**<cac:LineItem>**
3d0 3d1 3d2 3d3	00111111 (3f) 10000011 (83) 10000011 (83) 10000010 (82)	这些八位组是对 **BuyersID** 元素信息项的编码。 **[prefix]**属性，**[namespace name]**属性和**[local name]**属性都被索引为相关字符串，并已经都先于本信息项出现。**[local name]**属性被索引，是通过处理 **Order** 元素信息项的第一个孩子，名为 **BuyersID**，且其**[namespace name]**属性为"….Order:1:0"的元素信息项而实现的。 具有一个**[prefix]** 属性为"cac"（其索引为 4_{10}），一个**[namespace name]** 属性为"……..ateComponents:1:0"（索引为 4_{10}），以及一个**[local name]** 属性为"BuyersID"（索引为 3_{10}）的限定名字将被加入到 ELEMENT NAME 表中（其索引为 21_{10}）。	**<cac:BuyersID>**
3d4 3d5	10010000 (90) 01000001 (41)	这些八位组是 **BuyersID** 元素信息项中的 **character** 信息项的编码。 所处位置为 $3d4_{16}$ 的八位组，其值为 90_{16}，其中前两个比特（第一个到第二个比特）为'10'（标识），表示有一个元素信息项的孩子（即 **BuyersID** 元素信息项的孩子），且此孩子为一个 **character** 信息项的块（见 C.3.7.5）。第三个比特为'0'，表示提供了一个按照字面含义的字符串（见 C.15.3）。第四个比特为'1'，表示此按照字面含义的字符串应当被加入到 CONTENT CHARACTER CHUNK 表中（在本例中，小于 6_{10} 个字符的串被加入到 CONTENT CHARACTER CHUNK 表或 ATTRIBUTE VALUE 表中）。第五个和第六个比特都是'0'，表示该块的编码格式为 UTF-8（见 C.20.3.1）。第七个比特为'0'，表示被编码的 UTF-8 字符（**character** 信息项的块）的八位组的长度大于或等于 1_{10} 个八位组且小于或等于 2_{10} 个八位组，同时该长度在减去较低的界后，被编码为一个无符号整数（见 C.24.3.1），且放在第八个比特中。此无符号整数为 0_{10}，而长度为 1_{10}。 编码后的 UTF-8 字符的八位组所在的位置为第 41_{16} 个八位组。	**character** 信息项"A"
3d6	11110000 (f0)	本信息项是 **BuyersID** 元素信息项的终止符。	**</cac:BuyersID>**

附 件 E

对象标识符值的分配

（本附件不是本建议书 | 国际标准的组成部分）

下列对象标识符和对象描述符在本建议书 | 国际标准中分配：

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

"快速信息集 ASN.模块"

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

"快速信息集编码定义模块"

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

"快速信息集编码链接模块"

{joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) encodings(1) optional-xml-declaration(0)} – *在 A.1 中，被定义为 finf-doc-opt-decl*

"包含一个 XML 声明的一个快速信息集文档"

{joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0) encodings(1) no-xml-declaration(1)} -- *在 A.1 中，被定义为 finf-doc-no-decl*

"不包含 XML 声明的一个快速信息集文档"

参 考 资 料

[1] OASIS *Universal Business Language (UBL) 1.0.*

[2] IETF RFC 1952 (1996), *GZIP file format specification version 4.3.*

ITU-T 系列建议书

A系列	ITU-T工作的组织
D系列	一般资费原则
E系列	综合网络运行、电话业务、业务运行和人为因素
F系列	非话电信业务
G系列	传输系统和媒质、数字系统和网络
H系列	视听及多媒体系统
I系列	综合业务数字网
J系列	有线网络和电视、声音节目及其它多媒体信号的传输
K系列	干扰的防护
L系列	电缆和外部设备其它组件的结构、安装和保护
M系列	电信管理，包括TMN和网络维护
N系列	维护：国际声音节目和电视传输电路
O系列	测量设备的技术规范
P系列	电话传输质量、电话设施及本地线路网络
Q系列	交换和信令
R系列	电报传输
S系列	电报业务终端设备
T系列	远程信息处理业务的终端设备
U系列	电报交换
V系列	电话网上的数据通信
X系列	**数据网、开放系统通信和安全性**
Y系列	全球信息基础设施、互联网协议问题和下一代网络
Z系列	用于电信系统的语言和一般软件问题

瑞士印刷
2007年，日内瓦