
International Standard



8879

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION

Information processing — Text and office systems — Standard Generalized Markup Language (SGML)

Traitement de l'information — Systèmes bureautiques — Langage standard généralisé de balisage (SGML)

First edition — 1986-10-15

UDC 681.3.06

Ref. No. ISO 8879-1986 (E)

Descriptors : data processing, documentation, logical structure, programming (computers), artificial languages, programming languages.



Documents

Document Sharing Hub

Price based on 155 pages

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 8879 was prepared by Technical Committee ISO/TC 97, *Information processing systems*.

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

Contents

0 Introduction	1
0.1 Background	1
0.2 Objectives	2
0.3 Organization	3
1 Scope	4
2 Field of Application	4
3 References	5
4 Definitions	5
5 Notation	20
5.1 Syntactic Tokens	21
5.2 Ordering and Selection Symbols	21
6 Entity Structure	21
6.1 SGML Document	21
6.2 SGML Entities	21
6.2.1 S Separator	22
6.2.2 Entity End	22
6.2.3 Implied SGML Declaration	22
6.3 Non-SGML Data Entity	22
7 Element Structure	22
7.1 Prolog	22
7.2 Document Element	22
7.2.1 Limits	22
7.3 Element	22
7.3.1 Omitted Tag Minimization	22
7.3.1.1 Start-tag Omission	22
7.3.1.2 End-tag Omission	23
7.3.2 Data Tag Minimization	23
7.3.3 Quantities	23
7.4 Start-tag	23
7.4.1 Minimization	23
7.4.1.1 Empty Start-tag	23
7.4.1.2 Unclosed Start-tag	24
7.4.1.3 NET-enabling Start-tag	24
7.4.2 Quantities	24
7.5 End-tag	24
7.5.1 Minimization	24
7.5.1.1 Empty End-tag	24
7.5.1.2 Unclosed End-tag	24
7.5.1.3 Null End-tag	24
7.6 Content	24
7.6.1 Record Boundaries	25
7.7 Document Type Specification	25
7.7.1 General Entity References	25
7.7.2 Parameter Entity References	25
7.8 Generic Identifier (GI) Specification	26
7.8.1 Rank Feature	26
7.8.1.1 Full Generic Identifier	26
7.8.1.2 Rank Stem	26
7.9 Attribute Specification List	26

7.9.1	Minimization	26
7.9.1.1	Omitted Attribute Specification	26
7.9.1.2	Omitted Attribute Name	26
7.9.2	Quantities	26
7.9.3	Attribute Value Specification	26
7.9.3.1	Minimization	27
7.9.4	Attribute Value	27
7.9.4.1	Syntactic Requirements	27
7.9.4.2	Fixed Attribute	27
7.9.4.3	General Entity Name	27
7.9.4.4	Notation	27
7.9.4.5	Quantities	27
8	Processing Instruction	27
8.1	Quantities	28
9	Common Constructs	28
9.1	Replaceable Character Data	28
9.2	Character Data	28
9.2.1	SGML Character	28
9.2.2	Function Character	28
9.3	Name	28
9.3.1	Quantities	28
9.4	Entity References	28
9.4.1	Quantities	28
9.4.2	Limits	28
9.4.3	Obfuscatory Entity References	29
9.4.4	Named Entity Reference	29
9.4.5	Reference End	29
9.4.6	Short Reference	29
9.4.6.1	Equivalent Reference String	29
9.5	Character Reference	30
9.6	Delimiter Recognition	30
9.6.1	Recognition Modes	30
9.6.2	Contextual Constraints	31
9.6.3	Order of Recognition	32
9.6.4	Delimiters Starting with the Same Character	32
9.6.5	Short References with Blank Sequences	32
9.6.5.1	Quantities	32
9.6.6	Name Characters	32
9.7	Markup Suppression	32
9.8	Capacity	32
10	Markup Declarations: General	34
10.1	Parts of Declarations	34
10.1.1	Parameter Separator	34
10.1.2	Parameter Literal	34
10.1.2.1	Quantities	34
10.1.3	Group	34
10.1.3.1	Quantities	34
10.1.4	Declaration Separator	34
10.1.5	Associated Element Type	35
10.1.6	External Identifier	35
10.1.6.1	Quantities	35
10.1.6.2	Capacities	35
10.1.7	Minimum Literal	35
10.1.7.1	Quantities	35
10.2	Formal Public Identifier	35
10.2.1	Owner Identifier	35
10.2.1.1	ISO Owner Identifier	35
10.2.1.2	Registered Owner Identifier	35

10.2.1.3	Unregistered Owner Identifier	36
10.2.2	Text Identifier	36
10.2.2.1	Public Text Class	36
10.2.2.2	Public Text Description	36
10.2.2.3	Public Text Language	36
10.2.2.4	Public Text Designating Sequence	36
10.2.2.5	Public Text Display Version	37
10.3	Comment Declaration	37
10.4	Marked Section Declaration	37
10.4.1	Quantities	37
10.4.2	Status Keyword Specification	37
10.5	Entity Declaration	38
10.5.1	Entity Name	38
10.5.1.1	Quantities	38
10.5.1.2	Capacities	38
10.5.2	Entity Text	38
10.5.3	Data Text	38
10.5.4	Bracketed Text	39
10.5.4.1	Quantities	39
10.5.5	External Entity Specification	39
11	Markup Declarations: Document Type Definition	39
11.1	Document Type Declaration	39
11.2	Element Declaration	40
11.2.1	Element Type	40
11.2.1.1	Ranked Element	40
11.2.1.2	Quantities	40
11.2.2	Omitted Tag Minimization	40
11.2.3	Declared Content	40
11.2.4	Content Model	40
11.2.4.1	Connector	41
11.2.4.2	Occurrence Indicator	41
11.2.4.3	Ambiguous Content Model	41
11.2.4.4	Data Tag Group	41
11.2.4.5	Quantities	42
11.2.5	Exceptions	42
11.2.5.1	Inclusions	42
11.2.5.2	Exclusions	42
11.3	Attribute Definition List Declaration	42
11.3.1	Quantities	42
11.3.2	Attribute Name	42
11.3.3	Declared Value	43
11.3.4	Default Value	43
11.3.4.1	Quantities	43
11.3.4.2	Capacities	43
11.4	Notation Declaration	43
11.5	Short Reference Mapping Declaration	44
11.6	Short Reference Use Declaration	44
11.6.1	Use in Document Type Declaration	44
11.6.2	Use in Document Instance	44
11.6.3	Current Map	44
12	Markup Declarations: Link Process Definition	44
12.1	Link Type Declaration	44
12.1.1	Simple Link Specification	45
12.1.2	Implicit Link Specification	45
12.1.3	Explicit Link Specification	45
12.1.3.1	Limits	45
12.1.4	Link Type Declaration Subset	45
12.1.4.1	Parameter Entities	45
12.1.4.2	Link Attributes	45

12.1.4.3	Simple Link	45
12.2	Link Set Declaration	45
12.2.1	Source Element Specification	46
12.2.2	Result Element Specification	46
12.3	Link Set Use Declaration	46
12.3.1	Use in Link Type Declaration	46
12.3.2	Use in Document Instance	46
12.3.3	Current Link Set	46
13	SGML Declaration	47
13.1	Document Character Set	47
13.1.1	Character Set Description	47
13.1.1.1	Base Character Set	47
13.1.1.2	Described Character Set Portion	47
13.1.2	Non-SGML Character Identification	48
13.2	Capacity Set	48
13.3	Concrete Syntax Scope	48
13.4	Concrete Syntax	49
13.4.1	Public Concrete Syntax	49
13.4.2	Shunned Character Number Identification	49
13.4.3	Syntax-reference Character Set	50
13.4.4	Function Character Identification	50
13.4.5	Naming Rules	50
13.4.6	Delimiter Set	51
13.4.6.1	General Delimiters	51
13.4.6.2	Short Reference Delimiters	51
13.4.7	Reserved Name Use	51
13.4.8	Quantity Set	51
13.5	Feature Use	52
13.5.1	Markup Minimization Features	52
13.5.2	Link Type Features	52
13.5.3	Other Features	52
13.6	Application-specific Information	53
14	Reference and Core Concrete Syntaxes	53
15	Conformance	53
15.1	Conforming SGML Document	53
15.1.1	Basic SGML Document	53
15.1.2	Minimal SGML Document	53
15.1.3	Variant Conforming SGML Document	53
15.2	Conforming SGML Application	53
15.2.1	Application Conventions	53
15.2.2	Conformance of Documents	53
15.2.3	Conformance of Documentation	53
15.3	Conforming SGML System	53
15.3.1	Conformance of Documentation	54
15.3.2	Conformance to System Declaration	54
15.3.3	Support for Reference Concrete Syntax	54
15.3.4	Support for Reference Capacity Set	55
15.3.5	Consistency of Parsing	55
15.3.6	Application Conventions	55
15.4	Validating SGML Parser	56
15.4.1	Error Recognition	56
15.4.2	Identification of SGML Messages	56
15.4.3	Content of SGML Messages	56
15.5	Documentation Requirements	56
15.5.1	Standard Identification	56
15.5.2	Identification of SGML Constructs	56
15.5.3	Terminology	57
15.5.4	Variant Concrete Syntax	57

15.6	System Declaration	57
15.6.1	Concrete Syntaxes Supported	57
15.6.1.1	Concrete Syntax Changes	57
15.6.1.2	Character Set Translation	57
15.6.2	Validation Services	58

Annexes

A	Introduction to Generalized Markup	59
A.1	The Markup Process	59
A.2	Descriptive Markup	60
A.3	Rigorous Markup	62
A.4	Conclusion	64
A.5	Acknowledgments	65
A.6	Bibliography	65
B	Basic Concepts	66
B.1	Documents, Document Type Definitions, and Procedures	66
B.1.1	Documents	66
B.1.2	Document Type Definitions	66
B.1.3	Procedures	67
B.2	Markup	67
B.3	Distinguishing Markup from Text	68
B.3.1	Descriptive Markup Tags	68
B.3.2	Other Markup	69
B.3.3	Record Boundaries	69
B.3.3.1	Record Boundaries in Data	70
B.3.3.2	Record Boundaries in Markup	70
B.4	Document Structure	70
B.4.1	Document Type Definitions	70
B.4.2	Element Declarations	71
B.4.2.1	Content Models	71
B.4.2.2	Connectors and Occurrence Indicators	71
B.4.2.3	Entity References in Models	72
B.4.2.4	Name Groups	72
B.4.2.5	Data Characters	73
B.4.2.6	Empty Content	73
B.4.2.7	Non-SGML Data	73
B.4.2.8	Summary of Model Delimiters	74
B.5	Attributes	74
B.5.1	Specifying Attributes	74
B.5.1.1	Names	74
B.5.1.2	Attribute Values	75
B.5.2	Declaring Attributes	75
B.5.2.1	Attribute Definition Syntax	75
B.5.2.2	Complex Attribute Values	76
B.5.2.3	Name Token Groups	77
B.5.2.4	Changing Default Values	77
B.6	Entities	78
B.6.1	Entity Reference Syntax	78
B.6.2	Declaring Entities	78
B.6.2.1	Processing Instructions	79
B.6.2.2	Entities with Entity References	79
B.6.2.3	External Entities	79
B.6.2.4	Public Entities	80
B.7	Characters	80

B.7.1	Character Classification	80
B.7.2	Character References	81
B.7.3	Using Delimiter Characters as Data	82
B.8	Marked Sections	83
B.8.1	Ignoring a Marked Section	83
B.8.2	Versions of a Single Document	84
B.8.3	Unparsable Sections	84
B.8.4	Temporary Sections	85
B.8.5	Keyword Specification	85
B.8.6	Defining a Marked Section as an Entity	85
B.9	Unique Identifier Attributes	86
B.10	Content Reference Attributes	86
B.11	Content Model Exceptions	87
B.11.1	Included Elements	87
B.11.2	Excluded Elements	87
B.12	Document Type Declaration	88
B.13	Data Content	88
B.13.1	Data Content Representations	89
B.13.1.1	Character Data (PCDATA, CDATA, and RCDATA)	89
B.13.1.2	Non-SGML Data (NDATA)	89
B.13.2	Data Content Notations	90
B.13.2.1	Notations for Character Data	90
B.13.2.2	Notations for Non-SGML Data	91
B.13.2.3	Specifying Data Content Notations	91
B.14	Customizing	92
B.14.1	The SGML Declaration	92
B.14.1.1	Optional Features	92
B.14.1.2	Variant Concrete Syntax	92
B.14.2	Impact of Customization	92
B.15	Conformance	93
C	Additional Concepts	94
C.1	Markup Minimization Features	94
C.1.1	SHORTTAG: Tags With Omitted Markup	94
C.1.1.1	Unclosed Short Tags	95
C.1.1.2	Empty Tags	95
C.1.1.3	Attribute Minimization	95
C.1.2	OMITTAG: Tags May be Omitted	96
C.1.2.1	Tag Omission Concepts	97
C.1.2.2	Specifying Minimization	97
C.1.2.3	End-tag Omission: Intruding Start-tag	98
C.1.2.4	End-tag Omission: End-tag of Containing Element	98
C.1.2.5	Start-tag Omission: Contextually Required Element	99
C.1.2.6	Combination with Short Tag Minimization	99
C.1.2.7	Markup Minimization Considerations	99
C.1.3	SHORTREF: Short Reference Delimiters May Replace	
Complete Entity References		100
C.1.3.1	Typewriter Keyboarding: Generalized WYSIWYG	100
C.1.3.2	Typewriter Keyboarding Example: Defining a Short Reference Map	100
C.1.3.3	Typewriter Keyboarding Example: Activating a Short Reference Map	101
C.1.3.4	Tabular Matter Example	102
C.1.3.5	Special Requirements	103
C.1.4	DATATAG: Data May Also be a Tag	103
C.1.5	RANK: Ranks May be Omitted from Tags	106
C.2	LINK Features: SIMPLE, IMPLICIT, and EXPLICIT	107
C.2.1	Link Process Definitions	108
C.3	Other Features	108

C.3.1	CONCUR: Document Instances May Occur Concurrently	108
C.3.2	SUBDOC: Nested Subdocument Entities May Occur	109
C.3.3	FORMAL: Public Identifiers are Formal	109
D	Public Text	110
D.1	Element Sets	110
D.1.1	Common Element Types	110
D.1.2	Pro Forma Element Types	110
D.2	Data Content Notations	110
D.3	Variant Concrete Syntaxes	111
D.3.1	Multicode Concrete Syntaxes	111
D.4	Entity Sets	111
D.4.1	General Considerations	112
D.4.1.1	Format of Declarations	112
D.4.1.2	Corresponding Display Entity Sets	113
D.4.1.3	Entity Names	113
D.4.1.4	Organization of Entity Sets	114
D.4.2	Alphabetic Characters	114
D.4.2.1	Latin	114
D.4.2.2	Greek Alphabetic Characters	117
D.4.2.3	Cyrillic Alphabetic Characters	119
D.4.3	General Use	121
D.4.3.1	Numeric and Special Graphic Characters	121
D.4.3.2	Diacritical Mark Characters	123
D.4.3.3	Publishing Characters	123
D.4.3.4	Box and Line Drawing Characters	125
D.4.4	Technical Use	126
D.4.4.1	General	126
D.4.4.2	Greek Symbols	128
D.4.4.3	Alternative Greek Symbols	129
D.4.5	Additional Mathematical Symbols	130
D.4.5.1	Ordinary Symbols	130
D.4.5.2	Binary and Large Operators	130
D.4.5.3	Relations	131
D.4.5.4	Negated Relations	133
D.4.5.5	Arrow Relations	134
D.4.5.6	Opening and Closing Delimiters	135
E	Application Examples	136
E.1	Document Type Definition	136
E.2	Computer Graphics Metafile	140
E.3	Device-Independent Code Extension	140
E.3.1	Code Extension Facilities	140
E.3.1.1	Avoiding False Delimiter Recognition	141
E.3.1.2	Eliminating Device and Code Dependencies	143
F	Implementation Considerations	145
F.1	A Model of SGML Parsing	145
F.1.1	Physical Input	145
F.1.1.1	Entities	145
F.1.1.2	Record Boundaries	145
F.1.2	Recognition Modes	145
F.1.3	Markup Minimization	146
F.1.4	Translation	147
F.1.5	Command Language Analogy	147

- F.2 Initialization 147
 - F.2.1 Initial Procedure Mapping 147
 - F.2.2 Link Process Specification 147
 - F.2.3 Concurrent Document Instances 147
- F.3 Dynamic Procedure Mapping 148
- F.4 Error Handling 148
- G Conformance Classification and Certification 149
 - G.1 Classification Code 149
 - G.1.1 Feature Code 149
 - G.1.2 Validation Code 150
 - G.1.3 Syntax Code 151
 - G.2 Certification Considerations 151
- H Theoretical Basis for the SGML Content Model 152
 - H.1 Model Group Notation 152
 - H.2 Application of Automata Theory 152
 - H.3 Divergence from Automata Theory 153
- I Nonconforming Variations 154
 - I.1 Fixed-length Generic Identifiers 154
 - I.2 Single Delimiter 154

Figures

- 1 Character Classes: Abstract Syntax 29
- 2 Character Classes: Concrete Syntax 30
- 3 Reference Delimiter Set: General 31
- 4 Reference Delimiter Set: Short References 33
- 5 Reference Capacity Set 49
- 6 Reference Quantity Set 52
- 7 Reference Concrete Syntax 54
- 8 Typical SGML Declaration for Basic SGML Document 55
- 9 Element Markup 69
- 10 Start-tag with 2 Attributes 75
- 11 Multicode Basic Concrete Syntax 112
- 12 Graphics Metafile Attributes (1 of 2): Encoding and View 141
- 13 Graphics Metafile Attributes (2 of 2): Size and Rotation 142
- 14 Function Characters for Device-Independent Multicode Concrete Syntaxes 143

15	FSV Conformance Classification	150
----	--	-----

Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)

0 Introduction

This International Standard specifies a language for document representation referred to as the "Standard Generalized Markup Language" (SGML). SGML can be used for publishing in its broadest definition, ranging from single medium conventional publishing to multi-media data base publishing. SGML can also be used in office document processing when the benefits of human readability and interchange with publishing systems are required.

0.1 Background

A document can be viewed in the abstract as a structure of various types of element. An author organizes a book into chapters that contain paragraphs, for example, and figures that contain figure captions. An editor organizes a magazine into articles that contain paragraphs that contain words, and so on.

Processors treat these elements in different ways. A formatting program might print headings in a prominent type face, leave space between paragraphs, and otherwise visually convey the structure and other attributes to the reader. An information retrieval system would perhaps assign extra significance to words in a heading when creating its dictionary.

Although this connection between a document's attributes and its processing now seems obvious, it tended to be obscured by early text processing methods. In the days before automated typesetting, an editor would "mark up" a manuscript with the specific processing instructions that would create the desired format when executed by a compositor. Any connection between the instructions and the document's structure was purely in the editor's head.

Early computerized systems continued this approach by adding the process-specific "markup" to the machine-readable document file. The markup still consisted of specific processing instructions, but now they were in the language of a formatting program, rather than a human compositor. The file could not easily be used for a different purpose, or on a different computer system, without changing all the markup.

As users became more sophisticated, and as text processors became more powerful, approaches were developed that alleviated this problem. "Macro calls" (or "format calls") were used to identify points in the document where processing was to occur. The actual processing instructions were kept outside of the document, in "procedures" (or "macro definitions" or "stored formats"), where they could more easily be changed.

While the macro calls could be placed anywhere in a document, users began to recognize that most were placed at the start or end of document elements. It was natural, therefore, to choose names for such macros that were "generic identifiers" of the element types, rather than names that suggested particular processing (for example, "heading" rather than "format-17"), and so the practice of "generic coding" (or "generalized tagging") began.

Generic coding was a major step towards making automated text processing systems reflect the natural relationship between document attributes and processing. The advent of "generalized markup languages" in the early 1970's carried this trend further by providing a formal language base for generic coding. A generalized markup language observes two main principles:

0.1 Background

- a) Descriptive markup predominates and is distinguished from processing instructions.

Descriptive markup includes both generic identifiers and other attributes of document elements that motivate processing instructions. The processing instructions, which can be in any language, are normally collected outside of the document in procedures.

As the source file is scanned for markup and the various elements are recognized, the processing system executes the procedures associated with each element and attribute for that process. For other processes, different procedures can be associated with the same elements and attributes without changing the document markup.

When a processing instruction must be entered directly in a document, it is delimited differently from descriptive markup so that it can easily be located and changed for different processes.

- b) Markup is formally defined for each type of document.

A generalized markup language formalizes document markup by incorporating "document type definitions". Type definitions include a specification (like a formal grammar) of which elements and attributes can occur in a document and in what order. With this information it is possible to determine whether the markup for an individual document is correct (that is, complies with the type definition) and also to supply markup that is missing, because it can be inferred unambiguously from other markup that is present.

NOTE — A more detailed introduction to the concepts of generic coding and the Standard Generalized Markup Language can be found in annex A.

0.2 Objectives

The Standard Generalized Markup Language standardizes the application of the generic coding and generalized markup concepts. It provides a coherent and unambiguous syntax for describing whatever a user chooses to identify within a document. The language includes:

- An "abstract syntax" for descriptive markup of document elements.
- A "reference concrete syntax" that binds the abstract syntax to particular delimiter characters and quantities. Users can define

alternative concrete syntaxes to meet their requirements.

- Markup declarations that allow the user to define a specific vocabulary of generic identifiers and attributes for different document types.
- Provision for arbitrary data content. In generalized markup, "data" is anything that is not defined by the markup language. This can include specialized "data content notations" that require interpretation different from general text: formulas, images, non-Latin alphabets, previously formatted text, or graphics.
- Entity references: a non-system-specific technique for referring to content located outside the mainstream of the document, such as separately-written chapters, pi characters, photographs, etc.
- Special delimiters for processing instructions to distinguish them from descriptive markup. Processing instructions can be entered when needed for situations that cannot be handled by the procedures, but they can easily be found and modified later when a document is sent to a different processing system.

For a generalized markup language to be an acceptable standard, however, requires more than just providing the required functional capabilities. The language must have metalinguistic properties, in order to satisfy the constraints imposed by the need to use it in a multiplicity of environments. The major constraints, and the means by which the Standard Generalized Markup Language addresses them, can be summarized as follows:

- a) Documents "marked up" with the language must be processable by a wide range of text processing and word processing systems.

The full form of the language, with all optional features, offers generality and flexibility that can be exploited by sophisticated systems; less powerful systems need not support the features. To facilitate interchange between dissimilar systems, an "SGML declaration" describes any markup features or concrete syntax variations used in a document.

- b) The millions of existing text entry devices must be supported.

SGML documents, with the reference concrete syntax, can easily be keyboarded and understood by humans, without machine assistance. As a result:

- Use of SGML need not await the development and acceptance of a new generation of hardware — just software to process the documents on existing machines.
- Migration to such a new generation (when it comes) will be easier, as users will already be familiar with SGML.

- c) There must be no character set dependency, as documents might be keyed on a variety of devices.

The language has no dependency on a particular character set. Any character set that has bit combinations for letters, numerals, space, and delimiters is acceptable.

- d) There must be no processing, system, or device dependencies.

Generalized markup is predominantly descriptive and therefore inherently free of such dependencies. The occasional processing instruction is specially delimited so it can be found and converted for interchange, or when a different process renders the instruction irrelevant.

References to external parts of a document are indirect. The mappings to real system storage are made in "external entity declarations" that occur at the start of the document, where they can easily be modified for interchange.

The concrete syntax can be changed with the SGML declaration to accommodate any reserved system characters.

- e) There must be no national language bias.

The characters used for names can be augmented by any special national characters. Generic identifiers, attribute names, and other names used in descriptive markup are defined by the user in element and entity declarations.

The declaration names and keywords used in markup declarations can also be changed.

Multiple character repertoires, as used in multi-lingual documents, are supported.

- f) The language must accommodate familiar typewriter and word processor conventions.

The "short reference" and "data tag" capabilities support typewriter text entry conventions. Normal text containing paragraphs and quotations is interpretable as SGML although it is keyable with no visible markup.

- g) The language must not depend on a particular data stream or physical file organization.

The markup language has a virtual storage model in which documents consist of one or more storage entities, each of which is a sequence of characters. All real file access is handled by the processing system, which can decide whether the character sequence should be viewed as continuous, or whether it should reflect physical record boundaries.

- h) "Marked up" text must coexist with other data.

A processing system can allow text that conforms to this International Standard to occur in a data stream with other material, as long as the system can locate the start and end of the conforming text.

Similarly, a system can allow data content not defined by SGML to occur logically within a conforming document. The occurrence of such data is indicated by markup declarations to facilitate interchange.

- i) The markup must be usable by both humans and programs.

The Standard Generalized Markup Language is intended as a suitable interface for keyboarding and interchange without preprocessors. It allows extensive tailoring to accommodate user preferences in text entry conventions and the requirements of a variety of keyboards and displays.

However, it is recognized that many implementers will want to take advantage of the language's information capture capabilities to provide intelligent editing or to create SGML documents from a word processing front-end environment. SGML accommodates such uses by providing the following capabilities:

- Element content can be stored separately from the markup.
- Control characters can be used as delimiters.
- Mixed modes of data representation are permitted in a document.
- Multiple concurrent logical and layout structures are supported.

0.3 Organization

The organization of this International Standard is as follows:

- a) The physical organization of an SGML document as an entity structure is specified in clause 6.

- b) The logical organization of an SGML document as an element structure, and its representation with descriptive markup, is specified in clause 7.
- c) Processing instructions are discussed in clause 8.
- d) Common markup constructs, such as characters, entity references, and processing instructions, are covered in clause 9.
- e) Markup declarations with general applicability (comment, entity, and marked section) are specified in clause 10.
- f) Markup declarations that are used primarily to specify document type definitions (document type, element, notation, short reference mapping, and short reference use) are defined in clause 11.
- g) Markup declarations that are used primarily to specify link process definitions (link type, link attribute, link set, and link set use) are defined in clause 12.
- h) The SGML declaration, which specifies the document character set, capacity set, concrete syntax, and features, is defined in clause 13.
- i) The reference concrete syntax is defined in clause 14.
- j) Conformance of documents, applications, and systems is defined in clause 15.

There are also a number of annexes containing additional information; they are not integral parts of the body of this International Standard.

NOTE — This International Standard is a formal specification of a computer language, which may prove difficult reading for those whose expertise is in the production of documents, rather than compilers. Annexes A, B, and C discuss the main concepts in an informal tutorial style that should be more accessible to most readers. However, the reader should be aware that those annexes do not cover all SGML constructs, nor all details of those covered, and subtle distinctions are frequently ignored in the interest of presenting a clear overview.

1 Scope

This International Standard:

- a) Specifies an abstract syntax known as the Standard Generalized Markup Language (SGML). The language expresses the description of a document's structure and other attributes, as well as other information that makes the markup interpretable.

- b) Specifies a reference concrete syntax that binds the abstract syntax to specific characters and numeric values, and criteria for defining variant concrete syntaxes.
- c) Defines conforming documents in terms of their use of components of the language.
- d) Defines conforming systems in terms of their ability to process conforming documents and to recognize markup errors in them.
- e) Specifies how data not defined by this International Standard (such as images, graphics, or formatted text) can be included in a conforming document.

NOTE — This International Standard does not:

- a) Identify or specify "standard" document types, document architectures, or text structures.
- b) Specify the implementation, architecture, or markup error handling of conforming systems.
- c) Specify how conforming documents are to be created.
- d) Specify the data stream, message handling system, file structure, or other physical representation in which conforming documents are stored or interchanged, or any character set or coding scheme into or from which conforming documents might be translated for such purposes.
- e) Specify the data content representation or notation for images, graphics, formatted text, etc., that are included in a conforming document.

2 Field of Application

The Standard Generalized Markup Language can be used for documents that are processed by any text processing or word processing system. It is particularly applicable to:

- a) Documents that are interchanged among systems with differing text processing languages.
- b) Documents that are processed in more than one way, even when the procedures use the same text processing language.

Documents that exist solely in final formatted form are not within the field of application of this International Standard.

3 References

ISO 639, *Codes for the representation of names of languages*.¹⁾

ISO 646, *Information processing — 7-bit coded character set for information interchange*.

ISO 9069, *Information processing — SGML support facilities — SGML Document Interchange Format (SDIF)*.¹⁾

ISO 9070, *Information processing — SGML support facilities — Registration procedures for public text*.¹⁾

The following references are used in conjunction with illustrative material:

ISO 2022, *Information processing — ISO 7-bit and 8-bit coded character sets — Code extension techniques*.

ISO 3166, *Codes for the representation of names of countries*.

ISO 4873, *Information processing — ISO 8-bit code for information interchange — Structure and rules for implementation*.

ISO 6937, *Information processing — Coded character sets for text communication*.

ISO 8632/2, *Information processing systems — Computer graphics — Metafile for the storage and transfer of picture description information — Part 2: Character encoding*.¹⁾

ISO 8632/4, *Information processing systems — Computer graphics — Metafile for the storage and transfer of picture description information — Part 4: Clear text encoding*.¹⁾

4 Definitions

NOTE — The typographic conventions described in 5.1 are employed in this clause.

For the purposes of this International Standard, the following definitions apply:

4.1 abstract syntax (of SGML): Rules that define how markup is added to the data of a document, without regard to the specific characters used to represent the markup.

4.2 active document type (declaration): A document type declaration with respect to which an SGML entity is being parsed.

4.3 active link type (declaration): A link type declaration with respect to which an SGML entity is being parsed.

4.4 ambiguous content model: A *content model* for which an element or character string occurring in the document instance can satisfy more than one *primitive content token* without look-ahead.

NOTE — Ambiguous content models are prohibited in SGML.

4.5 application: Text processing application.

4.6 application convention: Application-specific rule governing the text of a document in areas that SGML leaves to user choice.

NOTE — There are two kinds: content conventions and markup conventions.

4.7 application-specific information: A parameter of the *SGML declaration* that specifies information required by an application and/or its architecture.

NOTE — For example, the information could identify an architecture and/or an application, or otherwise enable a system to determine whether it can process the document.

4.8 associated element type: An element type associated with the subject of a markup declaration by its *associated element type* parameter.

4.9 attribute (of an element): A characteristic quality, other than type or content.

4.10 attribute definition: A member of an attribute definition list; it defines an attribute name, allowed values, and default value.

4.11 attribute definition list: A set of one or more attribute definitions defined by the *attribute definition list* parameter of an attribute definition list declaration.

4.12 attribute (definition) list declaration: A markup declaration that associates an attribute definition list with one or more element types.

4.13 attribute list: Attribute specification list.

4.14 attribute list declaration: Attribute definition list declaration.

4.15 attribute specification: A member of an attribute specification list; it specifies the value of a single attribute.

4.16 attribute (specification) list: Markup that is a set of one or more attribute specifications.

¹⁾ At present at the stage of draft.

NOTE — Attribute specification lists occur in start-tags and link sets.

4.17 attribute value literal: A delimited character string that is interpreted as an *attribute value* by replacing references and ignoring or translating function characters.

4.18 available public text: Public text that is available to the general public, though its owner may require payment of a fee or performance of other conditions.

4.19 B sequence: An uninterrupted sequence of upper-case letter "B" characters; in a string assigned as a short reference, it denotes a blank sequence whose minimum length is the length of the B sequence.

4.20 base document element: A document element whose document type is the base document type.

4.21 base document type: The document type specified by the first document type declaration in a prolog.

4.22 basic SGML document: A conforming SGML document that uses the reference concrete syntax and capacity set and the SHORTTAG and OMITTAG markup minimization features.

NOTE — It also uses the SHORTREF feature by virtue of using the reference concrete syntax.

4.23 bit: Binary digit; that is, either zero or one.

4.24 bit combination: An ordered collection of bits, interpretable as a binary number.

4.25 blank sequence: An uninterrupted sequence of **SPACE** and/or **SEPCHAR** characters.

4.26 capacity: A named limit on some aspect of the size or complexity of a document, expressed as a number of points that can be accumulated for a kind of object or for all objects.

NOTE — The set of capacities is defined by the abstract syntax, but values are assigned to them by individual documents and SGML systems.

4.27 capacity set: A set of assignments of numeric values to capacity names.

NOTE — In an SGML declaration, the capacity set identifies the maximum capacity requirements of the document (its actual requirements may be lower). A capacity set can also be defined by an application, to limit the capacity requirements of documents that implementations of the application must process, or by a system, to specify the capacity requirements that it is capable of meeting.

4.28 CDATA: Character data.

4.29 CDATA entity: Character data entity.

4.30 chain of (link) processes: Processes, performed sequentially, that form a chain in which the source of the first process is an instance of the base document type, and the result of each process but the last is the source for the next. Any portion of the chain can be iterated.

NOTE — For example, a complex page makeup application could include three document types—logical, galley, and page—and two link processes—justification and castoff. The justification process would create an instance of a galley from an instance of a logical document, and the castoff process would in turn create pages from the galleys. The two processes could be iterated, as decisions made during castoff could require re-justification of the galleys at different sizes.

4.31 character: An atom of information with an individual meaning, defined by a character repertoire.

NOTES

1 There are two kinds: graphic character and control character.

2 A character can occur in a context in which it has a meaning, defined by markup or a data content notation, that supercedes or supplements its meaning in the character repertoire.

4.32 (character) class: A set of characters that have a common purpose in the abstract syntax, such as non-SGML characters or separator characters.

NOTE — Specific characters are assigned to character classes in four different ways:

- explicitly, by the abstract syntax (*Special*, *Digit*, *LC Letter*, and *UC Letter*);
- explicitly, by the concrete syntax (*LCNMSTRT*, *FUNCHAR*, *SEPCHAR*, etc.);
- implicitly, as a result of explicit assignments made to delimiter roles or other character classes (*DELMCHAR* and *DATACHAR*); or
- explicitly, by the document character set (*NONSGML*).

4.33 character data: Zero or more characters that occur in a context in which no markup is recognized, other than the delimiters that end the *character data*. Such characters are classified as data characters because they were declared to be so.

4.34 character data entity: An entity whose text is treated as *character data* when referenced and is not dependent on a specific system, device, or application process.

4.35 character entity set: A public entity set consisting of general entities that are graphic characters.

NOTES

1 Character entities are used for characters that have no coded representation in the document character set, or that cannot be keyboarded conveniently, or to achieve device independence for characters whose bit combinations do not cause proper display on all output devices.

2 There are two kinds of character entity sets: definitional and display.

4.36 character number: A *number* that represents the base-10 integer equivalent of the coded representation of a character, obtained by treating the sequence of bit combinations as a single base-2 integer.

4.37 character reference: A reference that is replaced by a single character.

NOTE — There are two kinds: named character reference and numeric character reference.

4.38 character repertoire: A set of characters that are used together. Meanings are defined for each character, and can also be defined for control sequences of multiple characters.

NOTE — When a character occurs in a control sequence, the meaning of the sequence supercedes the meanings of the individual characters.

4.39 character set: A mapping of a character repertoire onto a code set such that each character is associated with its coded representation.

4.40 (character) string: A sequence of characters.

4.41 class: Character class.

4.42 code extension: The use of a single coded representation for more than one character, without changing the document character set.

NOTE — When multiple national languages occur in a document, graphic repertoire code extension may be useful.

4.43 code set: A set of bit combinations of equal size, ordered by their numeric values, which must be consecutive.

NOTE — For example, a code set whose bit combinations have 8 bits (an "8-bit code") could consist of as many as 256 bit combinations, ranging in value from 00000000 through 11111111

(0 through 255 in the decimal number base), or it could consist of any contiguous subset of those bit combinations.

4.44 code set position: The numeric value of a bit combination in a code set.

4.45 coded representation: The representation of a character as a sequence of one or more bit combinations of equal size.

4.46 comment: A portion of a markup declaration that contains explanations or remarks intended to aid persons working with the document.

4.47 comment declaration: A markup declaration that contains only comments.

4.48 concrete syntax (of SGML): A binding of the abstract syntax to particular delimiter characters, quantities, markup declaration names, etc.

4.49 concrete syntax parameter: A parameter of the SGML declaration that identifies the concrete syntax used in document elements and (usually) prologs.

NOTE — The parameter consists of parameters that identify the syntax-reference character set, function characters, shunned characters, naming rules, delimiter use, reserved name use, and quantitative characteristics.

4.50 conforming SGML application: An SGML application that requires documents to be conforming SGML documents, and whose documentation meets the requirements of this International Standard.

4.51 conforming SGML document: An SGML document that complies with all provisions of this International Standard.

4.52 containing element: An element within which a subelement occurs.

4.53 content: Characters that occur between the start-tag and end-tag of an element in a document instance. They can be interpreted as data, proper subelements, included subelements, other markup, or a mixture of them.

NOTE — If an element has an explicit content reference, or its declared content is "EMPTY", the content is empty. In such cases, the application itself may generate data and process it as though it were content data.

4.54 content convention: An application convention governing data content, such as a restriction on length, allowable characters, or use of upper-case and lower-case letters.

NOTE — A content convention is essentially an informal data content notation, usually restricted to a single element type.

4.55 (content) model: Parameter of an element declaration that specifies the *model group* and *exceptions* that define the allowed *content* of the element.

4.56 content model nesting level: The largest number of successive *grpo* or *dtgo* delimiters that occur in a *content model* without a corresponding *grpc* or *dtgc* delimiter.

4.57 content reference (attribute): An impliable attribute whose value is referenced by the application to generate content data.

NOTE — When an element has an explicit content reference, the element's *content* in the document instance is empty.

4.58 contextual sequence: A sequence of one or more markup characters that must follow a delimiter string within the same entity in order for the string to be recognized as a delimiter.

4.59 contextually optional element: An element

- that can occur only because it is an inclusion; or
- whose *content token* in the currently applicable model group is a contextually optional token.

4.60 contextually optional token: A *content token* that

- is an inherently optional token; or
- has a **plus** occurrence indicator and has been satisfied; or
- is in a model group that is itself a contextually optional token, no tokens of which have been satisfied.

4.61 contextually required element: An element that is not a contextually optional element and

- whose *generic identifier* is the *document type name*; or
- whose currently applicable model token is a contextually required token.

NOTE — An element could be neither contextually required nor contextually optional; for example, an element whose currently applicable model token is in an **or** group that has no inherently optional tokens.

4.62 contextually required token: A *content token* that

- is the only one in its model group; or
- is in a **seq** group
 - that
 - is itself a contextually required token; or

- contains a token which has been satisfied;
- and

- all preceding tokens of which
 - have been satisfied; or
 - are contextually optional.

4.63 control character: A character that controls the interpretation, presentation, or other processing of the characters that follow it; for example, a tab character.

4.64 control sequence: A sequence of characters, beginning with a control character, that controls the interpretation, presentation, or other processing of the characters that follow it; for example, an escape sequence.

4.65 core concrete syntax: A variant of the reference concrete syntax that has no short reference delimiters.

4.66 corresponding content (of a content token): The element(s) and/or data in a document instance that correspond to a *content token*.

4.67 current attribute: An attribute whose current (that is, most recently specified) value is its default value.

NOTE — The start-tag cannot be omitted for the first occurrence of an element with a current attribute.

4.68 current element: The open element whose *start-tag* most recently occurred (or was omitted through markup minimization).

4.69 current link set: The link set associated with the current element by a *link set use declaration* in the element content or link type definition. If the current element has no associated link set, the previous current link set continues to be the current link set.

4.70 current map: The short reference map associated with the current element by a *short reference use declaration* in the element content or document type definition. If the current element has no associated map, the previous current map continues to be the current map.

4.71 current rank: A number that is appended to a rank stem in a tag to derive the generic identifier. For a *start-tag* it is the *rank suffix* of the most recent element with the identical *rank stem*, or a *rank stem* in the same *ranked group*. For an *end-tag* it is the *rank suffix* of the most recent open element with the identical *rank stem*.

4.72 data: The characters of a document that represent the inherent information content; characters that are not recognized as markup.

4.73 data character: An *SGML character* that is interpreted as data in the context in which it occurs, either because it was declared to be data, or because it was not recognizable as markup.

4.74 data content: The portion of an element's *content* that is data rather than markup or a subelement.

4.75 data content notation: An application-specific interpretation of an element's data content, or of a non-SGML data entity, that usually extends or differs from the normal meaning of the document character set.

NOTE — It is specified for an element's *content* by a notation attribute, and for a non-SGML data entity by the *notation name* parameter of the entity declaration.

4.76 data tag: A string that conforms to the data tag pattern of an open element. It serves both as the *end-tag* of the open element and as *character data* in the element that contains it.

4.77 data tag group: A model group token that associates a data tag pattern with a target element type.

NOTE — Within an instance of a target element, the data content and that of any subelements is scanned for a string that conforms to the pattern (a "data tag").

4.78 data tag pattern: A data tag group token that defines the strings that, if they occurred in the proper context, would constitute a data tag.

4.79 declaration: Markup declaration.

4.80 declaration subset: A delimited portion of a markup declaration in which other declarations can occur.

NOTE — Declaration subsets occur only in document type, link type, and marked section declarations.

4.81 declared concrete syntax: The concrete syntax described by the *concrete syntax* parameter of the *SGML declaration*.

4.82 dedicated data characters: Character class consisting of each *SGML character* that has no possible meaning as markup; a member is never treated as anything but a *data character*.

4.83 default entity: The entity that is referenced by a general entity reference with an undeclared name.

4.84 default value: A portion of an attribute definition that specifies the attribute value to be used if there is no *attribute specification* for it.

4.85 definitional (character) entity set: A character entity set whose purpose is to define entity names for graphic characters, but not actually to display them. Its *public identifier* does not include a *public text display version*.

NOTE — During processing, the system replaces a definitional entity set with a corresponding display character entity set for the appropriate output device.

4.86 delimiter characters: Character class that consists of each *SGML character*, other than a *name character* or *function character*, that occurs in a string assigned to a delimiter role by the concrete syntax.

4.87 delimiter-in-context: A character string that consists of a delimiter string followed immediately in the same entity by a contextual sequence.

4.88 delimiter role: A role defined by the abstract syntax, and filled by a character string assigned by the concrete syntax, that involves identifying parts of the markup and/or distinguishing markup from data.

4.89 delimiter set: A set of assignments of delimiter strings to the abstract syntax delimiter roles.

4.90 delimiter set parameter: A parameter of an SGML declaration that identifies the delimiter set used in the declared concrete syntax.

4.91 delimiter (string): A character string assigned to a delimiter role by the concrete syntax.

4.92 descriptive markup: Markup that describes the structure and other attributes of a document in a non-system-specific manner, independently of any processing that may be performed on it. In particular, it uses tags to express the element structure.

4.93 device-dependent version (of public text): Public text whose *formal public identifier* differs from that of another public text only by the addition of a *public text display version*, which identifies the display devices supported or coding scheme used.

4.94 digits: Character class composed of the 10 Arabic numerals from "0" through "9".

4.95 display (character) entity set: An entity set with the same entity names as a corresponding definitional character entity set, but which causes the characters to be displayed. It is a device-dependent version of the corresponding definitional entity set.

4.96 document: A collection of information that is processed as a unit. A document is classified as being of a particular document type.

NOTE — In this International Standard, the term almost invariably means (without loss of accuracy) an SGML document.

4.97 document architecture: Rules for the formulation of text processing applications.

NOTE — For example, a document architecture can define:

- a) attribute semantics for use in a variety of element definitions;
- b) element classes, based on which attributes the elements have;
- c) structural rules for defining document types in terms of element classes;
- d) link processes, and how they are affected by the values of attributes; and/or
- e) information to accompany a document during interchange (a "document profile").

4.98 document character set: The character set used for all markup in an SGML document, and initially (at least) for data.

NOTE — When a document is interchanged between systems, its character set is translated to the receiving system character set.

4.99 document element: The element that is the outermost element of an instance of a document type; that is, the element whose *generic identifier* is the *document type name*.

4.100 document instance: Instance of a document type.

4.101 document instance set: The portion of an SGML document entity or SGML subdocument entity in the entity structure that contains one or more instances of document types. It is co-extensive with the base document element in the element structure.

NOTE — When the concurrent instance feature is used, multiple instances can exist in a document, and data and markup can be shared among them.

4.102 document type: A class of documents having similar characteristics; for example, journal, article, technical manual, or memo.

4.103 (document) type declaration: A markup declaration that contains the formal specification of a document type definition.

4.104 document type declaration subset: The element, entity, and short reference sets occurring within the declaration subset of a document type declaration.

NOTE — The external entity referenced from the document type declaration is considered part of the declaration subset.

4.105 document (type) definition: Rules, determined by an application, that apply SGML to the markup of documents of a particular type. A document type definition includes a formal specification, expressed in a document type declaration, of the element types, element relationships and attributes, and references that can be represented by markup. It thereby defines the vocabulary of the markup for which SGML defines the syntax.

NOTE — A document type definition can also include comments that describe the semantics of elements and attributes, and any application conventions.

4.106 document type specification: A portion of a tag or entity reference that identifies the document type instances within which the tag or entity reference will be processed.

4.107 ds (separator): A declaration separator, occurring in declaration subsets.

4.108 DTD: Document type definition.

4.109 effective status (of a marked section): The highest priority status keyword specified on a marked section declaration.

4.110 element: A component of the hierarchical structure defined by a document type definition; it is identified in a document instance by descriptive markup, usually a start-tag and end-tag.

NOTE — An element is classified as being of a particular element type.

4.111 element declaration: A markup declaration that contains the formal specification of the part of an element type definition that deals with the content and markup minimization.

4.112 element set: A set of element declarations that are used together.

NOTE — An element set can be public text.

4.113 element structure: The organization of a document into hierarchies of elements, with each hierarchy conforming to a different document type definition.

4.114 element type: A class of elements having similar characteristics; for example, paragraph, chapter, abstract, footnote, or bibliography.

4.115 element (type) definition: Application-specific rules that apply SGML to the markup of elements of a particular type. An element type

definition includes a formal specification, expressed in element and attribute definition list declarations, of the content, markup minimization, and attributes allowed for a specified element type.

NOTE — An element type definition is normally part of a document type definition.

4.116 element type parameter: A parameter of an element declaration that identifies the type of element to which the definition applies.

NOTE — The specification can be direct, in the form of an individual *generic identifier* or member of a *name group*, or indirect, via a *ranked element* or member of a *ranked group*.

4.117 empty link set: A link set in which all result element types are implied and no attributes are specified.

4.118 empty map: A short reference map in which all delimiters are mapped to nothing.

NOTE — The empty map need not (and cannot) be declared explicitly, but can be referenced by its reserved name, which is "#EMPTY" in the reference concrete syntax.

4.119 end-tag: Descriptive markup that identifies the end of an element.

4.120 entity: A collection of characters that can be referenced as a unit.

NOTES

1 Objects such as book chapters written by different authors, pi characters, or photographs, are often best managed by maintaining them as individual entities.

2 The physical organization of entities is system-specific, and could take the form of files, members of a partitioned data set, components of a data structure, or entries in a symbol table.

4.121 entity declaration: A markup declaration that assigns an SGML name to an entity so that it can be referenced.

4.122 entity end (signal): A signal from the system that an entity's replacement text has ended.

4.123 entity manager: A program (or portion of a program or a combination of programs), such as a file system or symbol table, that can maintain and provide access to multiple entities.

4.124 entity reference: A reference that is replaced by an entity.

NOTE — There are two kinds: named entity reference and short reference.

4.125 entity set: A set of entity declarations that are used together.

NOTE — An entity set can be public text.

4.126 entity structure: The organization of a document into one or more separate entities.

NOTE — The first entity is an *SGML document entity*; it contains entity references that indicate where the other entities belong with respect to it.

4.127 entity text: The entity declaration parameter that specifies the replacement text, either by including it in a parameter literal, or by pointing to it with an external identifier.

4.128 equivalent reference string: A character string, consisting of an entity reference and possibly an *RE* and/or *RS*, that replaces a short reference when a document is converted from a concrete syntax that supports short references to one that does not.

4.129 escape sequence: A control sequence whose first character is escape (ESC).

4.130 exceptions: A parameter of an element declaration that modifies the effect of the element's *content model*, and the content models of elements occurring within it, by permitting inclusions and prohibiting exclusions.

4.131 exclusions: Elements that are not allowed anywhere in the content of an element or its subelements even though the applicable content model or inclusions would permit them optionally.

4.132 explicit content reference: A content reference that was specified in an *attribute specification*.

4.133 explicit link (process definition): A link process definition in which the result element types and their attributes and multiple sets of link attribute values can be specified.

4.134 external entity: An entity whose text is not incorporated directly in an entity declaration; its system identifier and/or public identifier is specified instead.

NOTE — A document type or link type declaration can include the identifier of an external entity containing all or part of the declaration subset; the external identifier serves simultaneously as both the entity declaration and the entity reference.

4.135 external identifier: A parameter that identifies an external entity or data content notation.

NOTE — There are two kinds: system identifier and public identifier.

4.136 fixed attribute: An attribute whose specified value (if any) must be identical to its default value.

4.137 formal public identifier: A public identifier that is constructed according to rules defined in this International Standard so that its owner identifier and the components of its text identifier can be distinguished.

4.138 formal public identifier error: An error in the construction or use of a *formal public identifier*, other than an error that would prevent it being a valid *minimum literal*.

NOTE — A formal public identifier error can occur only if “FORMAL YES” is specified on the SGML declaration. A failure of a *public identifier* to be a *minimum literal*, however, is always an error.

4.139 function character: A markup character, assigned by the concrete syntax, that can perform some SGML function in addition to potentially being recognized as markup. If it is not recognized as markup in a context in which data is allowed it is treated as data (unless the language dictates special treatment, as in the case of the **RE** and **RS** function characters).

4.140 function character identification parameter: A parameter of an SGML declaration that identifies the characters assigned to the **RE**, **RS**, and **SPACE** functions, and allows additional functions to be defined.

4.141 G0 set: In graphic repertoire code extension, the virtual character set that represents the document character set graphic characters whose character numbers are below 128, in their normal code set positions.

4.142 general delimiter (role): A delimiter role other than a short reference.

4.143 general entity: An entity that is referenced from within the content of an element or an attribute value literal.

4.144 general entity reference: A named entity reference to a general entity.

4.145 generic identifier: A name that identifies the element type of an element.

4.146 GI: Generic identifier.

4.147 graphic character: A character, such as a letter, digit, or punctuation, that normally occupies a single position when text is displayed.

4.148 graphic repertoire code extension: Code extension in which multiple graphic character sets are mapped onto positions of the document code set by using shift functions to invoke virtual character sets.

4.149 group: The portion of a parameter that is bounded by a balanced pair of **grpo** and **grpc** delimiters or **dtgo** and **dtgc** delimiters.

NOTE — There are five kinds: name group, name token group, model group, data tag group, and data tag template group. A name, name token, or data tag template group cannot contain a group, but a model group can contain a model group and a data tag group can contain a data tag template group.

4.150 ID: Unique identifier.

4.151 ID reference list: An attribute value that is a list of ID reference values.

4.152 ID reference value: An attribute value that is a *name* specified as an *id value* of an element in the same document instance.

4.153 ID value: An attribute value that is a *name* that uniquely identifies the element; that is, it cannot be the same as any other *id value* in the same document instance.

4.154 impliable attribute: An attribute for which there need not be an *attribute specification*, and whose value is defined by the application when it is not specified.

4.155 implicit link (process definition): A link process definition in which the result element types and their attributes are all implied by the application, but multiple sets of link attribute values can be specified.

4.156 inert function characters: Character class consisting of function characters whose additional SGML “function” is to do nothing.

4.157 included subelement: A subelement that is not permitted by its containing element’s model, but is permitted by an inclusion exception.

4.158 inclusions: Elements that are allowed anywhere in the content of an element or its subelements even though the applicable model does not permit them.

4.159 inherently optional token: A model group token that:

- a) has an **opt** or **rep** occurrence indicator; or
- b) is an **or** group, one of whose tokens is inherently optional; or
- c) is an **and** or **seq** group, all of whose tokens are inherently optional.

4.160 instance (of a document type): The data and markup for a hierarchy of elements that conforms to a document type definition.

4.161 interpreted parameter literal: The text of a *parameter literal*, exclusive of the literal delimiters, in which character and parameter entity references have been replaced.

4.162 ISO owner identifier: An *owner identifier*, consisting of an ISO publication number or character set registration number, that is used when a *public identifier* identifies, or is assigned by, an ISO publication, or identifies an ISO registered character set.

4.163 ISO text description: A *public text description*, consisting of the last element of an ISO publication title (without part designation, if any), that is used when a *public identifier* identifies an ISO publication.

4.164 keyword: A parameter that is a reserved name defined by the concrete syntax, as opposed to arbitrary text.

NOTE — In parameters where either a keyword or a name defined by an application could be specified, the keyword is always preceded by the reserved name indicator. An application is therefore able to define names without regard to whether those names are also used by the concrete syntax.

4.165 link attribute: An attribute of a source element type that is meaningful only in the context of a particular process that is performed on the source document instance.

4.166 link process: A process that creates a new instance of some document type (the result) from an existing instance of the same or another document type (the source). Processes can be chained, so that the result of one is the source for the next.

NOTE — Examples of link processes include editing, in which the source and result document types are usually the same, and formatting, in which they are usually different.

4.167 link process definition: Application-specific rules that apply SGML to describe a link process. A link process definition includes a formal specification, expressed in a *link type declaration*, of the link between elements of the source and result, including the definitions of source attributes applicable to the link process ("link attributes").

NOTES

1 A link process definition can also include comments that describe the semantics of the process, including the meaning of the link attributes and their effect on the process.

2 There are three kinds of link process definitions: simple, implicit, and explicit.

4.168 link set: A named set of associations, declared by a *link set declaration*, in which elements of the source document type are linked to elements of the result document type. For each element link, source link attributes and result element attributes can be specified.

4.169 link set declaration: A markup declaration that defines a link set.

4.170 link type declaration: A markup declaration that contains the formal specification of a link process definition.

4.171 link type declaration subset: The entity sets, link attribute sets, and link set and link set use declarations, that occur within the declaration subset of a link type declaration.

NOTE — The external entity referenced from the link type declaration is considered part of the declaration subset.

4.172 locking shift: A shift function that applies until another locking shift function occurs.

4.173 lower-case letters: Character class composed of the 26 unaccented small letters from "a" through "z".

4.174 lower-case name characters: Character class consisting of each additional lower-case *name character* assigned by the concrete syntax.

4.175 lower-case name start characters: Character class consisting of each additional lower-case *name start character* assigned by the concrete syntax.

4.176 LPD: Link process definition.

4.177 map: Short reference map.

4.178 mark up: To add markup to a document.

4.179 marked section: A section of the document that has been identified for a special purpose, such as ignoring markup within it.

4.180 marked section declaration: A markup declaration that identifies a marked section and specifies how it is to be treated.

4.181 marked section end: The closing delimiter sequence of a marked section declaration.

4.182 marked section start: The opening delimiter sequence of a marked section declaration.

4.183 markup: Text that is added to the data of a document in order to convey information about it.

NOTE — There are four kinds of markup: descriptive markup (tags), references, markup declarations, and processing instructions.

4.184 markup character: An SGML character that, depending on the context, could be interpreted either as markup or data.

4.185 markup convention: Application convention governing markup, such as a rule for the formulation of an entity name, or a preferred subset of allowed short reference delimiters.

4.186 (markup) declaration: Markup that controls how other markup of a document is to be interpreted.

NOTE — There are 13 kinds: SGML, entity, element, attribute definition list, notation, document type, link type, link set, link use, marked section, short reference mapping, short reference use, and comment.

4.187 (markup) minimization feature: A feature of SGML that allows markup to be minimized by shortening or omitting tags, or shortening entity references.

NOTE — Markup minimization features do not affect the document type definition, so a minimized document can be sent to a system that does not support these features by first restoring the omitted markup. There are five kinds: SHORTTAG, OMITTAG, SHORTREF, DATATAG, and RANK.

4.188 markup-scan-in characters: Character class consisting of function characters that restore markup recognition if it was suppressed by the occurrence of a markup-scan-out character.

4.189 markup-scan-out characters: Character class consisting of function characters that suppress markup recognition until the occurrence of a markup-scan-in character or entity end.

4.190 markup-scan-suppress characters: A character class consisting of function characters that suppress markup recognition for the immediately following character in the same entity (if any).

4.191 minimal SGML document: A conforming SGML document that uses the core concrete syntax throughout, no features, and the reference capacity set.

4.192 minimization feature: Markup minimization feature.

4.193 model: Content model.

4.194 model group: A component of a content model that specifies the order of occurrence of elements and character strings in an element's *content*, as modified by *exceptions* specified in the *content model* of the element and in the content models of other open elements.

4.195 multicode basic concrete syntax: A multicode variant of the basic concrete syntax in which markup is not recognized when code extension is in use.

4.196 multicode concrete syntax: A concrete syntax that allows code extension control characters to be SGML characters.

4.197 multicode core concrete syntax: A multicode variant of the core concrete syntax in which markup is not recognized when code extension is in use.

4.198 name: A name token whose first character is a name start character.

4.199 name character: A character that can occur in a name: name start characters, digits, and others designated by the concrete syntax.

4.200 name group: A group whose tokens are required to be names.

4.201 name start character: A character that can begin a name: letters, and others designated by the concrete syntax.

4.202 name token: A character string, consisting solely of name characters, whose length is restricted by the NAMELEN quantity.

NOTE — A name token that occurs in a group is also a token; one that occurs as an attribute value is not.

4.203 name token group: A group whose tokens are required to be name tokens.

4.204 named character reference: A character reference consisting of a delimited *function name*.

4.205 named entity reference: An entity reference consisting of a delimited name of a general entity or parameter entity (possibly qualified by a document type specification) that was declared by an entity declaration.

NOTE — A general entity reference can have an undeclared name if a default entity was declared.

4.206 naming rules parameter: A parameter of an SGML declaration that identifies additions to the

standard name alphabet character classes and specifies the case substitution.

4.207 non-SGML character: A character in the document character set whose coded representation never occurs in an SGML entity.

4.208 non-SGML data entity: An entity whose characters are not interpreted in accordance with this International Standard, and in which, therefore, no SGML markup can be recognized.

NOTE — The interpretation of a non-SGML data entity is governed by a data content notation, which may be defined by another International Standard.

4.209 NONSGML: The class of non-SGML characters, defined by the document character set.

4.210 normalized length (of an attribute specification list): A length calculated by ignoring the actual characters used for delimiting and separating the components and counting an extra fixed number per component instead.

4.211 notation attribute: An attribute whose value is a *notation name* that identifies the data content notation of the element's *content*.

NOTE — A notation attribute does not apply when there is an explicit content reference, as the element's *content* will be empty.

4.212 notation declaration: A markup declaration that associates a name with a notation identifier.

4.213 notation identifier: An *external identifier* that identifies a data content notation in a *notation declaration*. It can be a *public identifier* if the notation is public, and, if not, a description or other information sufficient to invoke a program to interpret the notation.

4.214 notation name: The name assigned to a data content notation by a notation declaration.

4.215 number: A name token consisting solely of digits.

4.216 number token: A name token whose first character is a digit.

NOTE — A number token that occurs in a group is also a token; one that occurs as an attribute value is not.

4.217 numeric character reference: A character reference consisting of a delimited *character number*.

4.218 object capacity: The capacity limit for a particular kind of object, such as entities defined or characters of entity text.

4.219 omitted tag minimization parameter: A parameter of an element declaration that specifies whether a technically valid omission of a start-tag or end-tag is considered a reportable markup error.

4.220 open element: An *element* whose *start-tag* has occurred (or been omitted through markup minimization), but whose *end-tag* has not yet occurred (or been omitted through markup minimization).

4.221 open entity: An entity that has been referenced but whose entity end has not yet occurred.

4.222 open marked section declaration: A marked section declaration whose *marked section start* has occurred but whose *marked section end* has not yet occurred.

4.223 owner identifier: The portion of a public identifier that identifies the owner or originator of public text.

NOTE — There are three kinds: ISO, registered, and unregistered.

4.224 parameter: The portion of a markup declaration that is bounded by parameter separators (whether required or optional). A parameter can contain other parameters.

4.225 parameter entity: An entity that is referenced from a markup declaration parameter.

4.226 parameter entity reference: A named entity reference to a parameter entity.

4.227 parameter literal: A parameter or token consisting of delimited replaceable parameter data.

4.228 parsed character data: Zero or more characters that occur in a context in which text is parsed and markup is recognized. They are classified as data characters because they were not recognized as markup during parsing.

4.229 PCDATA: Parsed character data.

4.230 PI entity: Processing instruction entity.

4.231 point: A unit of capacity measurement, roughly indicative of relative storage requirements.

4.232 procedure: Processing defined by an application to operate on elements of a particular type.

NOTES

1 A single procedure could be associated with more than one element type, and/or more than one procedure could operate on the same element type at different points in the document.

2 A procedure is usually part of a procedure set.

4.233 procedure set: The procedures that are used together for a given application process.

NOTE — In SGML applications, a procedure set usually constitutes the application processing for a link process definition.

4.234 processing instruction: Markup consisting of system-specific data that controls how a document is to be processed.

4.235 processing instruction entity: An entity whose text is treated as the *system data* of a *processing instruction* when referenced.

4.236 prolog: The portion of an SGML document or SGML subdocument entity that contains document type and link type declarations.

4.237 proper subelement: A subelement that is permitted by its containing element's model.

4.238 ps (separator): A parameter separator, occurring in markup declarations.

4.239 public identifier: A minimum literal that identifies public text.

NOTES

1 The public identifiers in a document can optionally be interpretable as formal public identifiers.

2 The system is responsible for converting public identifiers to system identifiers.

4.240 public text: Text that is known beyond the context of a single document or system environment, and which can be accessed with a public identifier.

NOTES

1 Examples are standard or registered document type definitions, entity sets, element sets, data content notations, and other markup constructs (see annex D).

2 Public text is not equivalent to published text; there is no implication of unrestricted public access. In particular, the owner of public text may choose to sell or license it to others, or to restrict its access to a single organization.

3 Public text simplifies access to shared constructs, reduces the amount of text that must be interchanged, and reduces the chance of copying errors.

4.241 public text class: The portion of a text identifier that identifies the SGML markup construct to which the public text conforms.

4.242 public text description: The portion of a text identifier that describes the public text.

4.243 public text designating sequence: The portion of a text identifier, used when public text is a character set, that contains an ISO 2022 escape sequence that designates the set.

4.244 public text display version: An optional portion of a text identifier that distinguishes among public text that has a common *public text description* by describing the devices supported or coding scheme used. If omitted, the public text is not device-dependent.

4.245 public text language: The portion of a text identifier that specifies the natural language in which the public text was written.

NOTE — It can be the language of the data, comments, and/or defined names.

4.246 quantity: A numeric restriction on some aspect of markup, such as the maximum length of a name or the maximum nesting level of open elements.

NOTE — Quantities are defined by the abstract syntax, but specific values are assigned to them by the concrete syntax.

4.247 quantity set: A set of assignments of numeric values to quantity names.

4.248 ranked element: An element whose *generic identifier* is composed of a *rank stem* and a *rank suffix*. When a ranked element begins, its *rank suffix* becomes the current rank for its *rank stem*, and for the rank stems in the *ranked group* (if any) of which the *rank stem* is a member.

4.249 ranked group: A group of rank stems that share the same current rank. When any ranked element whose stem is in the group begins, its *rank suffix* becomes the current rank for all rank stems in the group.

4.250 rank stem: A name from which a generic identifier can be derived by appending the current rank.

4.251 rank suffix: A number that is appended to a rank stem to form a *generic identifier*.

NOTE — The numbers are usually sequential, beginning with 1, so the resulting generic identifiers suggest the relative ranks of their elements (for example, H1, H2, and H3 for levels of heading elements, where “H” is the rank stem).

4.252 record: A division of an SGML entity, bounded by a record start and a record end character, normally corresponding to an input line on a text entry device.

NOTES

1 It is called a “record” rather than a “line” to distinguish it from the output lines created by a text formatter.

2 An SGML entity could consist of many records, a single record, or text with no record boundary characters at all (which can be thought of as being part of a record or without records, depending on whether record boundary characters occur elsewhere in the document).

4.253 record boundary (character): The record start (**RS**) or record end (**RE**) character.

4.254 record end: A *function character*, assigned by the concrete syntax, that represents the end of a record.

4.255 record start: A *function character*, assigned by the concrete syntax, that represents the start of a record.

4.256 reference: Markup that is replaced by other text, either an entity or a single character.

4.257 reference capacity set: The capacity set defined in this International Standard.

4.258 reference concrete syntax: A concrete syntax, defined in this International Standard, that is used in all SGML declarations.

4.259 reference delimiter set: The delimiter set, defined in this International Standard, that is used in the reference concrete syntax.

4.260 reference quantity set: The quantity set defined by this International Standard.

4.261 reference reserved name: A reserved name defined by this International Standard.

4.262 registered owner identifier: An owner identifier that was constructed in accordance with ISO 9070. It is unique among registered owner identifiers, and is distinguishable from ISO owner identifiers and unregistered owner identifiers.

4.263 replaceable character data: Character data in which a *general entity reference* or *character reference* is recognized and replaced.

NOTE — Markup that would terminate *replaceable character data* is not recognized in the replacement text of entities referenced within it.

4.264 replaceable parameter data: Character data in which a *parameter entity reference* or *character reference* is recognized and replaced.

NOTE — Markup that would terminate *replaceable parameter data* is not recognized in the replacement text of entities referenced within it.

4.265 replacement character: The character that replaces a *character reference*.

4.266 replacement text: The text of the entity that replaces an entity reference.

4.267 reportable markup error: A failure of a document to conform to this International Standard when it is parsed with respect to the active document and link types, other than:

- a) an ambiguous content model;
- b) an exclusion that could change a group’s required or optional status in a model;
- c) exceeding a capacity limit;
- d) an error in the SGML declaration;
- e) the occurrence of a non-SGML character; or
- f) an formal public identifier error.

4.268 required attribute: An attribute for which there must always be an *attribute specification* for the attribute value.

4.269 reserved name: A name defined by the concrete syntax, rather than by an application, such as a markup declaration name.

NOTE — Such names appear in this International Standard as syntactic literals.

4.270 reserved name use parameter: A parameter of the *SGML declaration* that specifies any replacement in the declared concrete syntax for a reference reserved name.

4.271 result document type (of a link): A document type, a new instance of which is created as the result of a link process.

4.272 result element type (of a link): An element that is defined in the result document type declaration.

4.273 s (separator): A separator, consisting of separator characters and other non-printing function characters, that occurs in markup and in *element content*.

4.274 satisfied token: A *content token* whose corresponding content has occurred.

4.275 SDATA entity: Specific character data entity.

4.276 separator: An *s*, *ds*, *ps*, or *ts*.

4.277 separator characters: A character class that consists of function characters that are allowed in separators and that will be replaced by **Space** in those contexts in which **RE** is replaced by **Space**.

4.278 SGML: Standard Generalized Markup Language

4.279 SGML application: Rules that apply SGML to a text processing application. An SGML application includes a formal specification of the markup constructs used in the application, expressed in SGML. It can also include a non-SGML definition of semantics, application conventions, and/or processing.

NOTES

1 The formal specification of an SGML application normally includes document type definitions, data content notations, and entity sets, and possibly a concrete syntax or capacity set. If processing is defined by the application, the formal specification could also include link process definitions.

2 The formal specification of an SGML application constitutes the common portions of the documents processed by the application. These common portions are frequently made available as public text.

3 The formal specification is usually accompanied by comments and/or documentation that explains the semantics, application conventions, and processing specifications of the application.

4 An SGML application exists independently of any implementation. However, if processing is defined by the application, the non-SGML definition could include application procedures, implemented in a programming or text processing language.

4.280 SGML character: A character that is permitted in an SGML entity.

4.281 SGML declaration: A markup declaration that specifies the character set, concrete syntax, optional features, and capacity requirements of a document's markup. It applies to all of the SGML entities of a document.

4.282 SGML document: A document that is represented as a sequence of characters, organized physically into an entity structure and logically into an element structure, essentially as described in this International Standard. An SGML document consists of data characters, which represent its information content, and markup

characters, which represent the structure of the data and other information useful for processing it. In particular, the markup describes at least one document type definition, and an instance of a structure conforming to the definition.

4.283 SGML document entity: The SGML entity that begins an SGML document. It contains, at a minimum, an SGML declaration, a base document type declaration, and the start and end (if not all) of a base document element.

4.284 SGML entity: An entity whose characters are interpreted as markup or data in accordance with this International Standard.

NOTE — There are three types of SGML entity: *SGML document entity*, *SGML subdocument entity*, and *SGML text entity*.

4.285 SGML parser: A program (or portion of a program or a combination of programs) that recognizes markup in conforming SGML documents.

NOTE — If an analogy were to be drawn to programming language processors, an SGML parser would be said to perform the functions of both a lexical analyzer and a parser with respect to SGML documents.

4.286 SGML subdocument entity: An SGML entity that conforms to the SGML declaration of the SGML document entity, while conforming to its own document type and link type declarations. It contains, at a minimum, a base document type declaration and the start and end of a base document element.

4.287 SGML system: A system that includes an SGML parser, an entity manager, and both or either of:

- a) an implementation of one or more SGML applications; and/or
- b) facilities for a user to implement SGML applications, with access to the SGML parser and entity manager.

4.288 SGML text entity: An SGML entity that conforms to the SGML declaration of the SGML document entity, and to the document type and link type declarations to which the entity from which it is referenced conforms.

4.289 shift function: In graphic repertoire code extension, a control sequence or control character that invokes a graphic character set.

NOTE — There are two kinds: single shift and locking shift.

4.290 short reference: Short reference string.

4.291 short reference delimiter role: A delimiter role to which zero or more strings can be assigned by the concrete syntax. When a short reference string is recognized, it is replaced by the general entity to whose name it is mapped in the current map, or is treated as a separator or data if it is mapped to nothing.

4.292 (short reference) map: A named set of associations, declared by a *short reference mapping declaration*, in which each short reference delimiter is mapped to a general entity name or to nothing.

4.293 short reference mapping declaration: A markup declaration that defines a short reference map.

4.294 short reference set: A set of short reference mapping, short reference use, and entity declarations that are used together.

NOTE — A short reference set can be public text.

4.295 short reference (string): A character string assigned to the short reference delimiter role by the concrete syntax.

4.296 short reference use declaration: A markup declaration that associates a short reference map with one or more element types, or identifies a new current map for the current element.

4.297 shunned character (number): A character number, identified by a concrete syntax, that should be avoided in documents employing the syntax because some systems might erroneously treat it as a control character.

4.298 significant SGML character: A *markup character* or *minimum data* character.

4.299 simple link (process definition): A link process definition in which the result element types and their attributes are all implied by the application, and only one set of link attribute values can be specified. The source document type must be the base.

4.300 single shift: A shift function that applies to the following character only.

4.301 source document type (of a link): A document type, an existing instance of which is the source of a link process.

4.302 source element type (of a link): An element type that is defined in the source document type declaration.

4.303 space: A *function character*, assigned by the concrete syntax, that represents a space.

4.304 specific character data entity: An entity whose text is treated as *character data* when referenced. The text is dependent on a specific system, device, or application process.

NOTE — A specific character data entity would normally be redefined for different applications, systems, or output devices.

4.305 Standard Generalized Markup Language: A language for document representation that formalizes markup and frees it of system and processing dependencies.

4.306 start-tag: Descriptive markup that identifies the start of an element and specifies its generic identifier and attributes.

4.307 status keyword: A marked section declaration parameter that specifies whether the marked section is to be ignored and, if not, whether it is to be treated as character data, replaceable character data, or normally.

4.308 string: Character string.

4.309 subelement: An element that occurs in the content of another element (the "containing element") in such a way that the subelement begins when the containing element is the current element.

4.310 syntax-reference character set: A character set, designated by a concrete syntax and known to all potential users of the syntax, that contains every significant SGML character. It enables a concrete syntax to be defined without regard to the particular document or system character sets with which it might be used.

4.311 system character set: The character set used in an SGML system.

4.312 system declaration: A declaration, included in the documentation for a conforming SGML system, that specifies the features, capacity set, concrete syntaxes, and character set that the system supports, the data content notations that it can interpret, and any validation services that it can perform.

4.313 system identifier: System data that specifies the file identifier, storage location, program invocation, data stream position, or other system-specific information that locates an external entity.

4.314 tag: Descriptive markup.

NOTE — There are two kinds: start-tag and end-tag.

4.315 target element: An element whose *generic identifier* is specified in a *data tag group*

4.316 text: Characters.

NOTE — The characters could have their normal character set meaning, or they could be interpreted in accordance with a data content notation as the representation of graphics, images, etc.

4.317 text identifier: The portion of a public identifier that identifies a public text so that it can be distinguished from any other public text with the same owner identifier.

NOTE — It consists of a *public text class*, an optional *unavailable text indicator*, a *public text description*, a *public text language*, and an optional *public text display version*.

4.318 text processing application: A related set of processes performed on documents of related types.

NOTE — Some examples are:

- a) Publication of technical manuals for a software developer: document types include installation, operation, and maintenance manuals; processes include creation, revision, formatting, and page layout for a variety of output devices.
- b) Preparation of manuscripts by independent authors for members of an association of publishers: document types include book, journal, and article; creation is the only defined process, as each publisher has its own methods of formatting and printing.
- c) Office correspondence: document types include memos, mail logs, and reports; processes include creation, revision, simple formatting, storage and retrieval, memo log update, and report generation.

4.319 token: The portion of a group, including a complete nested group (but not a *connector*), that is bounded by token separators (whether required or optional).

4.320 total capacity: A limit on the sum of all object capacities.

4.321 ts (separator): A token separator, occurring in groups.

4.322 type definition: Document type definition.

4.323 unavailable public text: Public text that is available only to a limited population, selected by its owner.

4.324 unique identifier: A *name* that uniquely identifies an element.

4.325 unregistered owner identifier: An owner identifier that can be distinguished from registered owner identifiers and ISO owner identifiers. As it is not constructed according to a registration standard, it could duplicate another unregistered owner identifier.

4.326 upper-case letters: Character class composed of the 26 capital letters from "A" through "Z".

4.327 upper-case name characters: Character class consisting of the upper-case forms of the corresponding lower-case name characters.

4.328 upper-case name start characters: Character class consisting of the upper-case forms of the corresponding lower-case name start characters.

4.329 validating SGML parser: A conforming SGML parser that can find and report a reportable markup error if (and only if) one exists.

4.330 variant concrete syntax: A concrete syntax other than the reference concrete syntax or core concrete syntax.

4.331 variant (conforming) SGML document: A conforming SGML document that uses a variant concrete syntax.

4.332 virtual character set: In graphic repertoire code extension, one of the character sets, known as G0, G1, G2, or G3, that represents the mapping of a real graphic character set, designated by an escape sequence, to a document code set position previously announced by an escape sequence.

5 Notation

NOTE — This clause describes the notation used in this International Standard to define the Standard Generalized Markup Language. This notation is not part of SGML itself (although there are some similarities between them), and therefore this clause should be considered as affecting only the presentation of this International Standard, not the substance.

The SGML abstract syntax is specified by formal syntax productions, each of which defines a "syntactic variable". A production consists of a reference number (in square brackets), the name of the syntactic variable being defined, an equals sign, and an expression that constitutes the definition.

[number] syntactic variable = expression

The expression is composed of one or more "syntactic tokens", parenthesized expressions, and symbols that define the ordering and selection among them.

5.1 Syntactic Tokens

The following list shows the syntactic token types using the typographic conventions employed for them in this International Standard, together with the source of their definitions:

syntactic variable. A syntactic token that is defined by a syntax production.

"SYNTACTIC LITERAL". A syntactic token consisting of a reserved name that is to be entered in markup exactly as it is shown in the syntax production, except that corresponding lower-case letters can be used if upper-case translation of general names is specified by the concrete syntax. Syntactic literals are defined whenever they occur in a syntax production, and the definition is applicable only in the context of that production.

delimiter role. A syntactic token that represents a delimiter string. Delimiter roles are defined in figure 3, which also lists the strings assigned to general delimiter roles by the reference concrete syntax. Strings assigned to the *shortref* delimiter role are shown in figure 4.

TERMINAL VARIABLE. A syntactic token that represents a character class whose members are not necessarily the same in all SGML documents. Terminal variables whose members are assigned by the concrete syntax are defined in figure 2. (The **NONSGML** variable, whose members are assigned by the document character set, is defined in 13.1.2.)

Terminal Constant. A syntactic token that represents either an entity end signal, or a character class whose members are the same in all SGML documents. Terminal constants are defined in figure 1.

5.2 Ordering and Selection Symbols

If there is more than one syntactic token in an expression, the ordering and selection among them is determined by symbols that connect them, as follows:

- , All must occur, in the order shown.
- & All must occur, in any order.
- | One and only one must occur.

Each selected syntactic token must occur once and only once unless the contrary is indicated by a suffix consisting of one of the following symbols:

- ? Optional (0 or 1 time).
- + Required and repeatable (1 or more times).
- * Optional and repeatable (0 or more times).

The occurrence suffixes are applied first, then the ordering connectors. Parentheses can be used as in mathematics to change these priorities.

6 Entity Structure

6.1 SGML Document

An *SGML document* is physically organized in an entity structure. The first entity is an *SGML document entity*; it contains entity references that indicate where the other entities belong with respect to it.

- [1] SGML document = *SGML document entity*,
(*SGML subdocument entity* |
SGML text entity | *non-SGML data entity*)*

NOTES

1 This International Standard does not constrain the physical organization of the document within the data stream, message handling protocol, file system, etc., that contains it. In particular, separate entities could occur in the same physical object, a single entity could be divided between multiple objects, and the objects could occur in any order.

2 This International Standard does not constrain the characters that can occur in a data stream outside of the SGML entities. Such characters would be interpreted according to applicable standards and conventions.

3 The SGML Document Interchange Format (SDIF) standardized in ISO 9069 allows an SGML document to be managed conveniently as a single object, while still preserving the entity structure.

6.2 SGML Entities

- [2] SGML document entity = *SGML declaration*,
*s**, *prolog*, *s**, *document instance set*, **Ee**

- [3] SGML subdocument entity = *prolog*, *s**,
document instance set, **Ee**

- [4] SGML text entity = *SGML character**, **Ee**

The characters of an SGML entity are parsed in accordance with this International Standard to recognize the *SGML declaration*, *prolog*, and *document instance set*, and their constituent parts.

Each *SGML character* is parsed in the order it occurs, in the following manner:

- a) The character is tested to see if it is part of a delimiter string (see 9.6). If a general delimiter string is recognized, the appropriate delimiter action is performed. If a *short reference* is recognized, it is treated as specified in 9.4.6.

6.2 SGML Entities

- b) If the character is not part of a delimiter string, or it is part of an unmapped short reference, it is tested to see if it is a separator; if so, it is ignored.
- c) If the character is not part of a delimiter or a separator, it is treated as data.

If an *SGML character* is a *function character*, its function is performed in addition to any other treatment it may receive.

6.2.1 S Separator

[5] $s = \text{SPACE} \mid \text{RE} \mid \text{RS} \mid \text{SEPCHAR}$

An *SGML character* is not considered part of an *s* if it can be interpreted as some other markup. If it is considered part of an *s* it is ignored.

6.2.2 Entity End

An *Ee* is an entity end signal.

NOTE — An *Ee* is not a character and is never treated as data. It can occur only where expressly permitted.

A system can represent an *Ee* in any manner that will allow it to be distinguished from SGML characters.

NOTE — For example, an *Ee* could be represented by the bit combination of a non-SGML character, if any have been assigned.

6.2.3 Implied SGML Declaration

While a document is processed exclusively by a single system, the system can imply the *SGML declaration*. The declaration must be present explicitly, however, if the document is subsequently sent to another system.

6.3 Non-SGML Data Entity

[6] non-SGML data entity = *character**, *Ee*

NOTE — A *non-SGML data entity* is declared with a notation parameter that identifies how the data is to be interpreted.

7 Element Structure

7.1 Prolog

[7] $\text{prolog} = \text{other prolog}^*,$
base document type declaration,
 (*document type declaration* |
other prolog)^{*}, (*link type declaration* |
other prolog)^{*}

[8] $\text{other prolog} = \text{comment declaration} \mid$
processing instruction | *s*

[9] $\text{base document type declaration} =$
document type declaration

An SGML document or subdocument entity conforms to the document type and link process definitions that are specified by the document type and link type declarations, respectively, in its *prolog*. An SGML text entity conforms to the *prolog* to which the entity from which it is referenced conforms. Parsing of an SGML entity is done with respect to those document and link types that are considered to be active.

NOTE — The system normally identifies the active document and link types to the SGML parser during initialization (see clause F.2).

Document type declarations in addition to the base are permitted only if "CONCUR YES" or "EXPLICIT YES" is specified on the *SGML declaration*.

7.2 Document Element

[10] $\text{document instance set}$
 = *base document element*

[11] $\text{base document element} = \text{document element}$

[12] $\text{document element} = \text{element}$

7.2.1 Limits

Instances of document type definitions other than the base are permitted only to the limit specified on the "CONCUR" parameter of the *SGML declaration*.

7.3 Element

[13] $\text{element} = \text{start-tag?}, \text{content}, \text{end-tag?}$

If an element has a *declared content* of "EMPTY", or an explicit content reference, the *end-tag* must be omitted.

NOTE — This requirement has nothing to do with markup minimization.

7.3.1 Omitted Tag Minimization

If "OMITTAG YES" is specified on the *SGML declaration*, a tag can be omitted as provided in this sub-sub-clause, as long as the omission would not create an ambiguity.

NOTE — A document type definition may consider a technically valid omission to be a markup error (see 11.2.2).

7.3.1.1 Start-tag Omission

The *start-tag* can be omitted if the element is a contextually required element and any other

elements that could occur are contextually optional elements, except if:

- a) the element type has a required attribute or *declared content*; or
- b) the *content* of the instance of the element is empty.

An omitted *start-tag* is treated as having an empty *attribute specification list*.

7.3.1.2 End-tag Omission

The *end-tag* can be omitted for a *document element*, or for an element that is followed either

- a) by the *end-tag* of another open *element*; or
- b) by an *element* or *SGML character* that is not allowed in its *content*.

NOTE — An element that is not allowed because it is an exclusion has the same effect as one that is not allowed because no token appears for it in the model group.

7.3.2 Data Tag Minimization

If “DATATAG YES” is specified on the *SGML declaration*, the *end-tag* can be omitted for an element that is the corresponding content of a data tag group.

The data content of the (target) element and its subelements is scanned for a string that conforms to one of the data tag templates in the element’s *data tag pattern*. That string, plus any succeeding characters that conform to the *data tag padding template*, are considered the element’s data tag. The data tag is treated both as the *end-tag* of the target element and as *character data* in its containing element.

NOTE — A *generic identifier* that occurs as a target element in a *data tag group* could also occur in other contexts as an *element token*. In those contexts, it would not be scanned for a data tag. It could also occur in other data tag groups, possibly with different data tag patterns.

The data content of the target element is scanned a character at a time. At each character, the longest possible *data tag template* in the *data tag pattern* is matched. If more than one target element is an open element, then at each character the templates of the most recently opened target element are tested first. If there is no match, the templates of the next most recently opened target element are tested, and so on.

NOTE — A data tag will therefore terminate the most recently opened target element whose data tag pattern it satisfies.

The matching of a *data tag pattern* to the content occurs after recognition of markup and replacement of references in the content, but before any *RE* or *RS* characters are ignored.

NOTE — The *data tag pattern* can therefore contain named character references to *RE* and *RS*.

A data tag cannot be recognized in any context in which end-tags are not recognized; for example, within a CDATA marked section.

In matching the *data tag padding template*, the template characters can be omitted, used once, or repeated indefinitely. The last (or only) use can stop short of the full template.

7.3.3 Quantities

The number of open elements cannot exceed the “TAGLVL” quantity.

The length of a data tag cannot exceed the “DTAGLEN” quantity.

7.4 Start-tag

[14] start-tag = (*stago*,
document type specification,
generic identifier specification,
attribute specification list, *s**, *tagc*) |
minimized start-tag

7.4.1 Minimization

[15] minimized start-tag = *empty start-tag* |
unclosed start-tag | *net-enabling start-tag*

A *start-tag* can be a *minimized start-tag* only if “SHORTTAG YES” is specified on the *SGML declaration*.

7.4.1.1 Empty Start-tag

[16] empty start-tag = *stago*,
document type specification, *s**, *tagc*

A *start-tag* can be an *empty start-tag* only if the *element* is in the base document instance, in which case the tag’s *generic identifier specification* is assumed to be:

- a) if “OMITTAG YES” is specified on the *SGML declaration*, the *generic identifier* of the most recently started open element in the base document type; or
- b) if “OMITTAG NO” is specified on the *SGML declaration*, the *generic identifier* of the most recently ended element in the base document type; or

7.4 Start-tag

- c) if there was no such previous applicable element, the *generic identifier* of the *document element*.

An *empty start-tag* is treated as having an empty *attribute specification list*.

7.4.1.2 Unclosed Start-tag

- [17] unclosed start-tag = **stago**,
document type specification,
generic identifier specification,
attribute specification list, s*

A *start-tag* can be an *unclosed start-tag* only if it is followed immediately by the character string assigned to the **stago** or **etago** delimiter role, regardless of whether the string begins a valid delimiter-in-context sequence.

7.4.1.3 NET-enabling Start-tag

- [18] net-enabling start-tag = **stago**,
document type specification,
generic identifier specification,
attribute specification list, s*, **net**

A *start-tag* can be a *net-enabling start-tag* only if its *element* is in the base document instance.

7.4.2 Quantities

The length of a *start-tag*, before resolution of references in the *attribute specification list* and exclusive of the **stago** and **tagc** or **net** delimiters, cannot exceed the "TAGLEN" quantity.

7.5 End-tag

- [19] end-tag = (**etago**,
document type specification,
generic identifier specification, s*, **tagc**) |
minimized end-tag

An *end-tag* ends the most recently started open *element*, within the instance of the document type specified by the *document type specification*, whose generic identifier is specified by the *generic identifier specification*.

7.5.1 Minimization

- [20] minimized end-tag = *empty end-tag* |
unclosed end-tag | *null end-tag*

An *end-tag* can be a *minimized end-tag* only if "SHORTTAG YES" is specified on the SGML *declaration*.

7.5.1.1 Empty End-tag

- [21] empty end-tag = **etago**, s*, **tagc**

If an *end-tag* is an *empty end-tag*, its *generic identifier* is that of the most recently started open element in the base document instance. If there was no such element, the *end-tag* is an error.

7.5.1.2 Unclosed End-tag

- [22] unclosed end-tag = **etago**,
document type specification,
generic identifier specification, s*

An *end-tag* can be an *unclosed end-tag* only if it is followed immediately by the character string assigned to the **stago** or **etago** delimiter role, regardless of whether the string begins a valid delimiter-in-context sequence.

7.5.1.3 Null End-tag

- [23] null end-tag = **net**

The **net** is recognized as a *null end tag* only if the *start-tag* of an open element in the base document type was a *net-enabling start-tag*. It is assumed to be the *end tag* of the most recently started such element.

7.6 Content

- [24] content = *mixed content* | *element content* |
replaceable character data | *character data*
- [25] mixed content = (*data character* | *element* |
other content)*
- [26] element content = (*element* | *other content* |
s)*
- [27] other content = *comment declaration* |
short reference use declaration |
link type use declaration |
processing instruction | **shortref** |
character reference |
general entity reference |
marked section declaration | **Ee**

An element's *declared content* or *content model* determines which of the four types of *content* it has, or whether it is empty, except that the *content* must be empty if the element has an explicit content reference.

The *content* of an element declared to be *character data* or *replaceable character data* is terminated only by an **etago** delimiter-in-context (which need not open a valid *end-tag*) or a valid **net**. Such termination is an error if it would have been an error had the *content* been *mixed content*.

NOTE — Content characters could be classed as data content for either of two reasons:

- a) Declared character data.

The element's entire content was declared to be *character data* or *replaceable character data* by the *declared content* parameter of the *element declaration*.

- b) Parsed character data.

The element was declared to have *mixed content*, and an *SGML character* within it was parsed as data because it was not recognized as markup.

7.6.1 Record Boundaries

If an **RS** in *content* is not interpreted as markup, it is ignored.

Within *content*, an **RE** remaining after replacement of all references and recognition of markup is treated as data unless its presence can be attributed solely to markup. That is:

- a) The first **RE** in an element is ignored if no **RS**, data, or proper subelement preceded it.
- b) The last **RE** in an element is ignored if no data or proper subelement follows it.
- c) An **RE** that does not immediately follow an **RS** or **RE** is ignored if no data or proper subelement intervened.

In applying these rules to an element, subelement content is ignored; that is, a proper or included subelement is treated as an atom that ends in the same record in which it begins.

NOTE — For example, in

```
record 1<outer><sub>
record 2</sub>
</outer>record 3
```

the first **RE** in the outer element is at the end of record 2. It is treated as data if "sub" is a proper subelement of "outer", but is ignored if "sub" is an included element, because no data or proper subelement would have preceded it in the outer element.

In either case, the first **RE** in the subelement is at the end of record 1; it is ignored because no data or proper subelement preceded it in the subelement.

An **RE** is deemed to occur immediately prior to the first data or proper subelement that follows it (that is, after any intervening markup declaration, processing instruction, or included subelement).

NOTES

1 A specific character data entity, non-SGML data entity, or SGML subdocument entity, is treated as data, while a processing instruction entity is not.

2 Although the handling of record boundaries is defined by SGML, there is no requirement that SGML documents must be organized in records.

3 No entity, including the *SGML document entity* and external entities, is deemed to start with an **RS** or end with an **RE** unless it really does.

7.7 Document Type Specification

[28] document type specification = *name group*?

Markup containing a *document type specification* is processed only if:

- a) a *name* in the *name group* is that of an active document type; or
- b) there is no *name group* (that is, the *document type specification* is empty).

NOTE — An effect of this requirement is that markup with an empty document type specification (that is, no *name group*) will apply to all document instances (or the only one).

A *name group* can be specified only if "CONCUR YES" or "EXPLICIT YES" is specified on the *SGML declaration*.

7.7.1 General Entity References

A general entity must have been declared in the *document type declaration* of each active document type named in the *document type specification* or, if the specification was empty or the entity was not so declared, in the base *document type declaration*.

NOTE — An effect of this requirement is that a general entity declared in the base document type (including the default entity) can be referenced in an instance of any document type in which no general entity with the same name was defined.

7.7.2 Parameter Entity References

A *name group* cannot be specified for a *parameter entity reference* within a *document type declaration* or *link type declaration*. Such an entity must have been declared within the same *document type declaration* or, in the case of the *link type declaration*, within the source *document type declaration*.

7.8 Generic Identifier (GI) Specification

[29] generic identifier specification =
generic identifier | *rank stem*

[30] generic identifier = *name*

A *generic identifier* is valid only if it was specified as an *element type* in the document type definition and:

- a) it was named in the *content model* of the *element declaration* for the element in which it occurred; or
- b) it was named in an applicable *inclusions exception*; or
- c) it is the *document type name* and it occurred in the *start-tag* or *end-tag* of the document element.

7.8.1 Rank Feature

If "RANK YES" is specified on the *SGML declaration*, the provisions of this sub-sub-clause apply.

7.8.1.1 Full Generic Identifier

If the full *generic identifier* is specified for a ranked element, its *rank suffix* becomes the current rank for its *rank stem*, and for the rank stems in any *ranked group* of which the *rank stem* is a member.

7.8.1.2 Rank Stem

A *generic identifier specification* can be a *rank stem* if it was declared via a *ranked element* or member of a *ranked group* in an applicable *element declaration*.

Specifying a *rank stem* is equivalent to specifying the *generic identifier* that is derived by appending the current rank to it. It is an error to specify a *rank stem* if no element previously occurred to establish the current rank for that stem.

7.9 Attribute Specification List

[31] attribute specification list =
*attribute specification**

[32] attribute specification = *s**, (*name*, *s**, *vi*, *s**)?, *attribute value specification*

The validity of the attribute specification list is determined by the *attribute definition list* associated with the element. If there is no associated *attribute definition list*, the *attribute specification list* must be empty.

Every attribute for which there is an *attribute definition*, other than an impliable attribute, must be specified (unless markup minimization is used, as described in 7.9.1.1).

There can be only one *attribute specification* for each *attribute definition*.

The leading *s* can only be omitted from an *attribute specification* that follows a delimiter.

7.9.1 Minimization

7.9.1.1 Omitted Attribute Specification

If "SHORTTAG YES" or "OMITTAG YES" is specified on the *SGML declaration*:

- a) There need be an *attribute specification* only for a required attribute, and for a current attribute on the first occurrence of any element in whose *attribute definition list* it appears. Other attributes will be treated as though specified with an *attribute value* equal to the declared *default value*.
- b) If there is an *attribute value specification* for a current attribute, the specified *attribute value* will become the default value. The new default affects all elements associated with the *attribute definition list* in which the attribute was defined.

7.9.1.2 Omitted Attribute Name

If "SHORTTAG YES" is specified on the *SGML declaration*, the *name* and *vi* can be omitted if the *attribute value specification* is an undelimited *name token* that is a member of a group specified in the declared *value* for that attribute.

NOTE — A *name token* can occur in only one group in an *attribute definition list* (see 11.3.3)

7.9.2 Quantities

The normalized length of the *attribute specification list* is the sum of the normalized lengths of each attribute name and attribute value specified, which cannot exceed the "ATTSPLEN" quantity.

The normalized length of an attribute name is the "NORMSEP" quantity plus the number of characters in the name.

7.9.3 Attribute Value Specification

[33] attribute value specification =
attribute value | *attribute value literal*

[34] attribute value literal = (*lit*,
*replaceable character data**, *lit*) | (*lita*,
*replaceable character data**, *lita*)

An *attribute value literal* is interpreted as an *attribute value* by replacing references within it, ignoring **Ee** and **RS**, and replacing an **RE** or **SEPCHAR** with a **SPACE**.

7.9.3.1 Minimization

If "SHORTTAG YES" is specified on the SGML declaration, an *attribute value specification* can be an *attribute value* (that is, not an *attribute value literal*) provided that it contains nothing but name characters.

7.9.4 Attribute Value

[35] attribute value = *character data* |
 general entity name | *id value* |
 id reference value | *id reference list* | *name*
 | *name list* | *name token* | *name token list* |
 notation name | *number* | *number list* |
 number token | *number token list*

[36] id value = *name*

[37] id reference list = *name list*

[38] id reference value = *name*

[39] name list = *name*, (s + , *name*)*

[40] name token list = *name token*, (s + ,
 name token)*

[41] notation name = *name*

[42] number list = *number*, (s + , *number*)*

[43] number token list = *number token*, (s + ,
 number token)*

The *declared value* parameter of an *attribute definition* determines which of the fourteen types of *attribute value* must be specified.

7.9.4.1 Syntactic Requirements

An *attribute value* must conform to the *declared value*.

If the *declared value* includes a group, the *attribute value* must be a token in that group.

An empty *attribute value literal* can be specified only for an *attribute value* whose type is *character data*.

7.9.4.2 Fixed Attribute

The specified *attribute value* for a fixed attribute must be its *default value*.

7.9.4.3 General Entity Name

The value of a *general entity name* attribute must be the name of an SGML subdocument entity or non-SGML data entity. If the attribute was specified in a *start-tag*, the entity definition must apply to each document instance in which the tag occurs. If the

attribute is a link attribute, the definition must apply to the source document type.

NOTE — The notation of a non-SGML data entity must apply to the same document types as the entity.

7.9.4.4 Notation

The *attribute value* of a notation attribute must be a *notation name* that was declared in the same document type declaration as the element.

It is an error to specify a value for a notation attribute if there is an explicit content reference.

NOTE — As the element's *content* will be empty, it is pointless to specify a notation for it. Even if the content reference were to a non-SGML data entity, the applicable notation would be specified by the *notation name* parameter of the entity declaration.

7.9.4.5 Quantities

The normalized length of an *attribute value*, whether specified directly or interpreted from an *attribute value literal*, is the "NORMSEP" quantity plus:

- for an *id reference list*, *name list*, *name token list*, *number list*, or *number token list* (even if there is only one token in the list), the sum of the number of characters in each token in the list, plus the "NORMSEP" quantity for each token in the list; or
- for all others, the number of characters in the value, plus the "NORMSEP" quantity for each reference to a "CDATA" or "SDATA" entity.

The normalized length of an *attribute value* cannot exceed the "LITLEN" quantity.

In a single *start tag*, the total number of names in *id reference value* and *id reference list* attribute values, whether defaulted or specified, cannot exceed the "GRPCNT" quantity.

8 Processing Instruction

[44] processing instruction = *pio*, *system data*,
 pic

[45] system data = *character data*

Processing instructions are deprecated, as they reduce portability of the document. If a processing instruction must be used, it should be defined as an entity, so that the *system data* will be confined to the *prolog*, where a recipient of the document can more easily locate and modify it.

A *processing instruction* that returns data must be defined as an "SDATA" entity and entered with an entity reference. One that does not return data should be defined as a "PI" entity.

No markup is recognized in system data other than the delimiter that would terminate it.

NOTE — The characters allowed in *system data* and their interpretation are defined by the system. If it is desired to allow non-SGML characters or the *pic* delimiter character in *system data*, an alternative way of entering them should be provided so that the actual characters do not occur in the document.

8.1 Quantities

The length of a *processing instruction*, exclusive of its delimiters, cannot exceed the "PILEN" quantity.

9 Common Constructs

9.1 Replaceable Character Data

[46] replaceable character data =
(*data character* | *character reference* |
general entity reference | **Ee**)*

Markup that would terminate *replaceable character data* is not recognized in an entity that was referenced from within the same *replaceable character data*.

An **Ee** can occur in *replaceable character data* only if the reference to the entity it terminates occurred in the same *replaceable character data*.

9.2 Character Data

[47] character data = *data character**

[48] data character = SGML character

[49] character = SGML character | **NONSGML**

NOTE — A non-SGML character can be entered as a data character within an SGML entity by using a *character reference*.

9.2.1 SGML Character

[50] SGML character = *markup character* |
DATACHAR

[51] markup character = *name character* |
function character | **DELMCHAR**

[52] name character = *name start character* |
Digit | **LCNMCHAR** | **UCNMCHAR**

[53] name start character = **LC Letter** | **UC Letter** |
LCNMSTR | **UCNMSTR**

9.2.2 Function Character

[54] function character = **RE** | **RS** | **SPACE** |
SEPCHAR | **MSOCHAR** | **MSICCHAR** |
MSSCHAR | **FUNCHAR**

9.3 Name

[55] name = *name start character*,
*name character**

[56] number = **Digit** +

[57] name token = *name character* +

[58] number token = **Digit**, *name character**

The upper-case form of each character in a *name*, *name token*, *number*, or *number token*, as specified by the "NAMECASE" parameter of the SGML *declaration*, is substituted for the character actually entered.

9.3.1 Quantities

The length of a *name*, *name token*, *number*, or *number token*, cannot exceed the "NAMELEN" quantity.

9.4 Entity References

The replacement text of an entity reference must comply with the syntactic and semantic requirements that govern the context of the reference. For purposes of this rule, a reference to an SGML *subdocument entity* or a non-SGML *data entity* is treated like a reference to a *data character*.

NOTE — Such entities can also be accessed by a *general entity name attribute*.

A reference to an undeclared entity is an error unless there is an applicable default entity (see 9.4.4).

A reference to an entity that has already been referenced and has not yet ended is invalid (i.e., entities cannot be referenced recursively).

9.4.1 Quantities

The number of open entities (except for the unreferenced SGML *document entity* in which the *document* begins) cannot exceed the "ENTLVL" quantity.

9.4.2 Limits

The number of open SGML subdocument entities cannot exceed the quantity specified on the "SUBDOC" parameter of the SGML *declaration*.

Variable	Characters	Numbers	Description
Digit	0 - 9	48 - 57	Digits
Ee	(system signal; not a character)		Entity end signal
LC Letter	a - z	97 - 122	Lower-case letters
Special	' () + , - . / : = ?	39 - 41 43 - 47 58 61 63	Special minimum data characters
UC Letter	A - Z	65 - 90	Upper-case letters

Figure 1 — Character Classes: Abstract Syntax

9.4.3 Obfuscatory Entity References

Any use of entity references that obscures the markup is deprecated.

NOTE — Most such abuses are prohibited by the syntax of SGML. The following principles should be observed (those that say “must” are restatements of syntax rules, stated formally elsewhere, that enforce the principles):

- The opening delimiter of a tag, *processing instruction*, declaration, literal, or other delimited text, must be in the same entity as the closing delimiter. An entity must not end in delimited text unless it began there, and an entity that begins there must end there.
- The content of an element or marked section that was declared to be *character data* or *replaceable character data* must (and other elements should) start and end in the same entity.
- An element's *start-tag* and *end-tag* (and a marked section's *marked section start* and *marked section end*) should be in the same entity, or they should be the replacement text of entities whose references are in the same entity.
- In a markup declaration, a reference must be replaced either by zero or more consecutive complete parameters (with any intervening *ps* separators), or, within a group, by one or more consecutive complete tokens (with any intervening *ts* separators and/or connectors).

9.4.4 Named Entity Reference

[59] general entity reference = **ero**,
document type specification, name,
reference end

[60] parameter entity reference = **pero**,
document type specification, name,
reference end

An entity *name* is required to be declared by an *entity declaration* before it can be used in a named entity reference, except that an undeclared *name* can be used in a *general entity reference* to refer to the default entity, if one was defined.

9.4.5 Reference End

[61] reference end = (**refc** | **RE**)?

NOTE — Ending a reference with an **RE** has the effect of suppressing the record end.

The **refc** or **RE** can be omitted only if the reference is not followed by a *name character*, or by a character that could be interpreted as the omitted *reference end*.

9.4.6 Short Reference

If a *short reference* is mapped to a general entity name in the current map, it is treated as markup and replaced by the named entity. If the *short reference* is mapped to nothing, each character in the delimiter string is treated as an *s* separator if it can be recognized as such, and if not it is treated as data.

9.4.6.1 Equivalent Reference String

A short reference can be removed from a document by replacing it with an equivalent reference string that contains a named entity reference. The entity *name* must be that to which the *short reference* is mapped in the current map.

If the short reference contains any quantity of **RS** or **RE** characters, the equivalent reference string will include a single **RS** or **RE**, or both, as shown in the following list:

Short Reference	Equivalent Reference String
No RS or RE	ero , name, refc
RS ; no RE	RS , ero , name, refc
RE ; no RS	ero , name, RE
Both RS and RE	RS , ero , name, RE

NOTES

1 Equivalent reference strings are used when a document is converted from a concrete syntax that supports short references to one that does not.

Variable	Characters	Numbers	Description
DATACHAR	(implicit)	(implicit)	Dedicated data characters
DELMCHAR	(implicit)	(implicit)	Delimiter characters
FUNCHAR	(none)	(none)	Inert function characters
LCNMCHAR	- .	45 46	Lower-case name characters
LCNMSTRT	(none)	(none)	Lower-case name start characters
MSICHAR	(none)	(none)	Markup-scan-in characters
MSOCHAR	(none)	(none)	Markup-scan-out characters
MSSCHAR	(none)	(none)	Markup-scan-suppress characters
RE		13	Record end character
RS		10	Record start character
SEPCHAR		9	Separator characters
SPACE		32	Space character
UCNMCHAR	- .	45 46	Upper-case name characters
UCNMSTRT	(none)	(none)	Upper-case name start characters

Figure 2 — Character Classes: Concrete Syntax

2 A single **RS** and/or **RE** is preserved in the equivalent reference string to prevent records from becoming joined, and possibly exceeding a system's maximum length restriction. They are not recognized as data: the **RS** because it never is, and the **RE** because it serves as the *reference end*.

9.5 Character Reference

[62] character reference = **cro**, (*function name* | *character number*), *reference end*

[63] function name = "RE" | "RS" | "SPACE" | *name*

[64] character number = *number*

A *name* specified as a *function name* must have been specified as an *added function* in the concrete syntax.

A *character reference* should be used when a character could not otherwise be entered conveniently in the text.

A replacement character is considered to be in the same entity as its reference.

A replacement character is treated as though it were entered directly except that the replacement for a numeric *character reference* is always treated as data.

NOTES

1 A system can determine its own internal representation for a replacement character. Care should be taken to distinguish a

normal *function character* (entered directly or as a replacement for a named character reference) from one that replaces a numeric character reference.

2 When a document is translated to a different document character set, the *character number* of each numeric character reference must be changed to the corresponding *character number* of the new set.

9.6 Delimiter Recognition

A delimiter string is recognized as satisfying a delimiter role only within the particular recognition mode (or modes) in which the role is meaningful and, in some cases, only if a contextual constraint is satisfied. The roles, their recognition modes, and any contextual constraints on their recognition, are listed in figure 3. Also shown are the strings assigned to general delimiters in the reference delimiter set, and the character numbers of those strings in the *translation-reference character set* of the reference concrete syntax. The strings assigned as short reference delimiters in the reference delimiter set are shown in figure 4.

9.6.1 Recognition Modes

The recognition modes are:

Mode	Meaning
CON	Recognized in <i>content</i> and in the <i>marked section</i> of marked section declarations that occur in <i>content</i> .
CXT	Recognized as part of the contextual sequence of a "CON" or "DS" mode delimiter-in-context. (See below.)

Name	String	Number	Mode	Constraint	Description of Role
AND	&	38	GRP		And connector
COM	--	45 45	CXT MD		Comment start or end
CRO	&#	38 35	CON LIT	CREF	Character reference open
DSC	]	93	DS MD		Declaration subset close
DSO	[	91	CXT MD		Declaration subset open
DTGC	]	93	GRP		Data tag group close
DTGO	[	91	GRP		Data tag group open
ERO	&	38	CON LIT	NMS	Entity reference open
ETAGO	</	60 47	CON	GI	End-tag open
GRPC	)	41	GRP		Group close
GRPO	(	40	CXT GRP MD		Group open
LIT	"	34	GRP LIT MD TAG		Literal start or end
LITA	'	39	GRP LIT MD TAG		Literal start or end (alternative)
MDC	>	62	CXT MD		Markup declaration close
MDO	<!	60 33	CON DS	DCL	Markup declaration open
MINUS	-	45	MD		Minus; exclusion
MSC	]]	93 93	CON DS	MSE	Marked section close
NET	/	47	CON TAG	ELEM	Null end-tag
OPT	?	63	GRP		Optional occurrence indicator
OR		124	GRP		Or connector
PERO	%	37	DS GRP MD	NMS	Parameter entity reference open
PIC	>	62	PI		Processing instruction close
PIO	<?	60 63	CON DS		Processing instruction open
PLUS	+	43	GRP MD		Required and repeatable; inclusion
REFC	;	59	REF		Reference close
REP	*	42	GRP		Optional and repeatable
RNI	#	35	GRP MD		Reserved name indicator
SEQ	,	44	GRP		Sequence connector
SHORTREF			CON		Short reference (see figure 4)
STAGO	<	60	CON TAG	GI	Start-tag open
TAGC	>	62	CXT TAG		Tag close
VI	=	61	TAG		Value indicator

Figure 3 — Reference Delimiter Set: General

- DS** Recognized in a declaration subset and in the *marked section* of marked section declarations that occur in a declaration subset.
- GRP** Recognized in a group.
- LIT** Recognized in a literal.
- MD** Recognized in a *markup declaration* (other than in a group or declaration subset).
- PI** Recognized in a *processing instruction*.
- REF** Recognized in a *general entity reference*, *parameter entity reference*, or *character reference*.
- TAG** recognized in a *start-tag* or *end-tag*.

9.6.2 Contextual Constraints

The most common constraint is that the delimiter string must start a delimiter-in-context in which it is followed by one of the listed contextual sequences:

- CREF** *name start character* or **Digit**
- DCL** *name start character*, **com**, **dso**, or **mdc**
- GI** *name start character*, or, if "SHORTTAG YES" is specified on the *SGML declaration*, **tagc**, or, if "CONCUR YES" is specified on the *SGML declaration*, **grp**
- MSE** **mdc**
- NMS** *name start character*, or, if "CONCUR YES" is specified on the *SGML declaration*, **grp**

Another contextual constraint is:

- ELEM** In "CON" mode, recognized only within an element whose *start-tag* was a *net-enabling start-tag*; in "TAG" mode, no constraints.

9.6.3 Order of Recognition

Delimiter strings (including any required contextual sequences) are recognized in the order they occur, with no overlap.

NOTE — For example, if “abc” and “bcd” are delimiter strings, and the document contains “abcde”, then “abc” will be recognized and parsing will continue at “d”, so “bcd” will not be recognized.

This rule holds true even if the recognized delimiter is semantically incorrect, or is a short reference that is mapped to nothing.

NOTE — For example, in the reference delimiter set, the solidus (/) will be recognized as part of an *etago* delimiter-in-context rather than as the *net*, even if the *end-tag* GI was not declared, or is not the GI of an open element.

9.6.4 Delimiters Starting with the Same Character

If multiple delimiter strings start with the same character, only the longest delimiter string or delimiter-in-context among them will be recognized at a given point in the document.

NOTE — For example, if “ab” and “abc” are delimiters and the document contains “abcd”, then “abc” will be recognized and parsing will continue at “d”, so “ab” will not be recognized.

This rule holds true even if the longer delimiter is semantically incorrect, or is a short reference that is mapped to nothing.

NOTE — If, in the previous example, “ab” and “abc” were short reference delimiters, short reference “abc” alone would be recognized and short reference “ab” would not be, even if short reference “abc” were mapped to nothing in the current map and short reference “ab” were mapped to an entity.

9.6.5 Short References with Blank Sequences

If there is a B sequence in the definition of a short reference delimiter, it will cause recognition of a blank sequence in the *content*. The minimum length of the blank sequence is the length of the B sequence.

NOTE — That is, one B means one or more blanks, two B's means two or more blanks, etc.

A string that could be recognized as more than one delimiter will be considered the string of the most specific delimiter that it satisfies.

NOTE — For example, a tab character would be recognized as “&#TAB;” in preference to “B”, and three spaces would be recognized as “BBB” in preference to “BB”.

9.6.5.1 Quantities

The length of a blank sequence recognized as a short reference cannot exceed the “BSEQLen” quantity. If an actual blank sequence is longer, only the first “BSEQLen” characters will be included in the short reference string, and parsing will resume with the following character.

9.6.6 Name Characters

If a *name start character* is assigned to a delimiter role, it will be recognized as a delimiter (in preference to recognition as a *name start character*) if a *name token* has not already begun; if found within a *name token*, it will be treated as a *name character*.

If general upper-case substitution is specified by the “NAMECASE” parameter of the *SGML declaration*, then for purposes of delimiter recognition, a *name character* assigned to a delimiter role is treated as though it were its upper-case form.

9.7 Markup Suppression

An *MSOCHAR* suppresses recognition of markup until an *MSICCHAR* or entity end occurs. An *MSSCHAR* does so for the next character in the same entity (if any).

NOTE — An *MSOCHAR* occurring in *character data* or other delimited text will therefore suppress recognition of the closing delimiter. An *MSSCHAR* could do so if it preceded the delimiter.

If markup recognition has not been suppressed by an *MSOCHAR*, an *MSICCHAR* has no effect on markup recognition, but is not an error.

If markup recognition has been suppressed by an *MSOCHAR*, a subsequent *MSOCHAR* or *MSSCHAR* has no effect on markup recognition, but is not an error.

An *MSOCHAR* that follows an *MSSCHAR* has no effect on markup recognition.

9.8 Capacity

The size and complexity of a document must not exceed the number of capacity points allowed by the document capacity set for the objects occurring in the document.

The names of the total and individual capacities, together with the values assigned to them in the reference capacity set, are shown in figure 5. The set's public identifier is:

ISO 8879-1986//CAPACITY Reference//EN

String	Number	Description
&#TAB;	9	Horizontal tab
&#RE;	13	Record end
&#RS;	10	Record start
&#RS;B	10 66	Leading blanks
&#RS;&#RE;	10 13	Empty record
&#RS;B&#RE;	10 66 13	Blank record
B&#RE;	66 13	Trailing blanks
&#SPACE;	32	Space
BB	66 66	Two or more blanks
"	34	Quotation mark
#	35	Number sign
%	37	Percent sign
'	39	Apostrophe
(	40	Left parenthesis
)	41	Right parenthesis
*	42	Asterisk
+	43	Plus sign
,	44	Comma
-	45	Hyphen
--	45 45	Two hyphens
:	58	Colon
;	59	Semicolon
=	61	Equals sign
@	64	Commercial at
[	91	Left square bracket
]	93	Right square bracket
^	94	Circumflex accent
_	95	Low line
{	123	Left curly bracket
	124	Vertical line
}	125	Right curly bracket
~	126	Tilde

Figure 4 — Reference Delimiter Set: Short References

The points accumulated for each type of object cannot exceed the value of that object's individual capacity, and the total points for all objects cannot exceed the "TOTALCAP" value.

The capacity values must be sufficient for the greatest capacity requirement among the possible sets of concurrent instances or chains of link processes that could be processed at once.

Points are counted for the SGML document entity and SGML text entities referenced from it, plus the set of possible open subdocument entities and SGML text entities referenced from them that would require the largest capacity.

NOTE — As an example of capacity calculation, when a concrete syntax with 32 short references and a "NAMELEN" of 8 is used, a capacity of 30818 or more would be required to accommodate a

document with 100 entities averaging 70 characters (800+7000), 200 element types (1600) with 2000 tokens in content models (16000) and 25 exceptions groups with a total of 50 names (200+400), 50 attributes with default values averaging 20 characters (400+1000) and 100 attribute name tokens (800), 5 data content notations with identifiers averaging 50 characters (40+250), 50 ID and 50 IDREF attributes (400+400), 5 short reference maps ($5 \times (8 + (8 \times 32)) = 1320$), and a single implicit link type declaration with 4 link sets, each containing 5 source element names (40+168).

10 Markup Declarations: General

NOTE — The declaration names and keywords in the syntax productions are reference reserved names that can be redefined for a variant concrete syntax with the *reserved name use* parameter of the *SGML declaration*.

10.1 Parts of Declarations

10.1.1 Parameter Separator

[65] $ps = s \mid \mathbf{Ee} \mid \textit{parameter entity reference} \mid \textit{comment}$

A *parameter entity reference* can be used wherever a parameter could occur. The entity must consist of zero or more of the consecutive complete parameters that follow the *ps* in which the reference occurs, together with any intervening *ps* separators. The entity must end within the same declaration.

An **Ee** can occur in a *ps* only if the reference to the entity it terminates occurs in a *ps* in the same declaration.

A required *ps* that is adjacent to a delimiter or another *ps* can be omitted if no ambiguity would be created thereby.

A *ps* must begin with an *s* if omitting it would create an ambiguity.

10.1.2 Parameter Literal

[66] $\textit{parameter literal} = (\textit{lit}, \textit{replaceable parameter data}, \textit{lit}) \mid (\textit{lita}, \textit{replaceable parameter data}, \textit{lita})$

[67] $\textit{replaceable parameter data} = (\textit{data character} \mid \textit{character reference} \mid \textit{parameter entity reference} \mid \mathbf{Ee})^*$

A *parameter literal* is interpreted as a parameter (or token) by replacing references while the declaration is being processed.

Except for parameter entity and character references, no markup is recognized in a *parameter literal* other than the terminal *lit* or *lita*, and those are not recognized within the replacement text of a reference.

NOTE — If the literal is in the *entity text* parameter of an *entity declaration*, markup characters in its text could be recognized when the entity is referenced.

An **Ee** can occur in *replaceable parameter data* only if the reference to the entity it terminates occurred in the same *replaceable parameter data*.

10.1.2.1 Quantities

The length of an interpreted *parameter literal* cannot exceed the “LITLEN” quantity (unless some other restriction is applied in the context in which it is used).

10.1.3 Group

[68] $\textit{name token group} = \textit{grpo}, ts^*, \textit{name token}, (ts^*, \textit{connector}, ts^*, \textit{name token})^*, ts^*, \textit{grpc}$

[69] $\textit{name group} = \textit{grpo}, ts^*, \textit{name}, (ts^*, \textit{connector}, ts^*, \textit{name})^*, ts^*, \textit{grpc}$

[70] $ts = s \mid \mathbf{Ee} \mid \textit{parameter entity reference}$

Only one type of *connector* should be used in a single *name group* or *name token group*.

NOTE — No specific type of *connector* is mandated, so that a group defined in an entity can be referenced as both a *model group* (where the specific connector is meaningful) and a *name group* (where it is not).

A token can occur only once in a single *name group* or *name token group*.

A *parameter entity reference* can be used anywhere in a group that a token could occur. The entity must consist of one or more of the consecutive complete tokens that follow the *ts* in which the reference occurs in the same group (i.e., at the same nesting level), together with any surrounding or intervening *ts* separators and any intervening connectors. The entity must end within the same group.

An **Ee** can occur in a *ts* only if:

- the *ts* follows a token (as opposed to a *connector*, *grpo*, or *dtgo*); and
- the reference to the entity the **Ee** terminates occurs in the same group (i.e., at the same nesting level).

10.1.3.1 Quantities

The number of tokens in a group cannot exceed the “GRPCNT” quantity.

10.1.4 Declaration Separator

[71] $ds = s \mid \mathbf{Ee} \mid \textit{parameter entity reference} \mid \textit{comment declaration} \mid \textit{processing instruction} \mid \textit{marked section declaration}$

A *parameter entity reference* in a *ds* must refer to an entity that consists of zero or more complete markup declarations and/or *ds* separators.

An **Ee** can occur in a *ds* only if the reference to the entity it terminates occurs in a *ds* in the same parameter.

10.1.5 Associated Element Type

[72] associated element type = *generic identifier* | *name group*

Each *name* in the *name group* must be a *generic identifier*.

10.1.6 External Identifier

[73] external identifier = ("SYSTEM" | ("PUBLIC", *ps* + , *public identifier*)), (*ps* + , *system identifier*)?

[74] public identifier = *minimum literal*

[75] system identifier = (*lit*, *system data*, *lit*) | (*lita*, *system data*, *lita*)

The *system identifier* can be omitted if the system can generate it from the *public identifier* and/or other information available to it.

If "FORMAL YES" is specified on the SGML declaration, a *public identifier* is interpreted as a *formal public identifier* (see 10.2) and a formal public identifier error can occur.

NOTE — It is still a *minimum literal*, and all requirements pertaining to minimum literals apply.

10.1.6.1 Quantities

The length of a *system identifier*, exclusive of delimiters, cannot exceed the "LITLEN" quantity.

10.1.6.2 Capacities

The number of characters of *entity text* counted towards the ENTCHCAP capacity for an *external identifier* is that of its *system identifier* component, whether specified or generated (and exclusive of delimiters).

10.1.7 Minimum Literal

[76] minimum literal = (*lit*, *minimum data*, *lit*) | (*lita*, *minimum data*, *lita*)

[77] minimum data = *minimum data character**

[78] minimum data character = **RS** | **RE** | **SPACE** | **LC Letter** | **UC Letter** | **Digit** | **Special**

A *minimum literal* is interpreted by ignoring **RS** and replacing a sequence of one or more **RE** and/or **SPACE** characters with a single **SPACE**.

10.1.7.1 Quantities

The length of an interpreted *minimum literal*, exclusive of delimiters, cannot exceed the "LITLEN" quantity of the reference quantity set, regardless of the concrete syntax used.

10.2 Formal Public Identifier

[79] formal public identifier = *owner identifier*, "///", *text identifier*

A *formal public identifier* cannot contain consecutive solidi ("///") except where expressly permitted by this sub-clause.

NOTE — As a *public identifier* is a *minimum literal*, **RS** characters are removed, and sequences of one or more **RE** and/or **SPACE** characters replaced by a single **SPACE**, prior to interpretation as a *formal public identifier*. The *minimum literal* length limitation applies to the interpreted text (see 10.1.7.1).

10.2.1 Owner Identifier

[80] owner identifier = *ISO owner identifier* | *registered owner identifier* | *unregistered owner identifier*

NOTE — In formulating an *owner identifier*, standards such as ISO 3166 can be helpful.

10.2.1.1 ISO Owner Identifier

[81] ISO owner identifier = *minimum data*

The usual form of *ISO owner identifier* can be used only when the *public identifier* identifies an ISO publication or is assigned within one. It consists of the ISO publication number, without the language suffix.

NOTE — For example, the *ISO owner identifier* for public text defined in this document is "ISO 8879-1986" in all translations. If the public text is translated, that fact is indicated by specifying the appropriate *public text language* in the *text identifier*.

A special form of *ISO owner identifier* can be used only when the public text is an ISO registered character set and the *public text class* is "CHARSET". It consists of the string "ISO Registration Number ", followed by the registration number of the character set.

10.2.1.2 Registered Owner Identifier

[82] registered owner identifier = "+///", *minimum data*

NOTE — A registered owner identifier could be a citation of a national or industry standard, or some other unique identifier assigned in accordance with ISO 9070.

10.2 Formal Public Identifier

10.2.1.3 Unregistered Owner Identifier

[83] unregistered owner identifier = "-//",
minimum data

NOTE — An unregistered owner identifier could be a (presumably unique) designation created by a trade organization or other user community, or by an individual.

10.2.2 Text Identifier

[84] text identifier = *public text class*, **SPACE**,
unavailable text indicator?,
public text description, "//",
(*public text language* |
public text designating sequence), ("//",
public text display version)?

[85] unavailable text indicator = "-//"

If the *unavailable text indicator* is present, the text is unavailable public text; otherwise, it is available public text.

If the *public text class* is "CHARSET", the *text identifier* includes a *public text designating sequence*; otherwise, it includes a *public text language*.

A *text identifier* cannot be the same as another *text identifier* in a *formal public identifier* that has the same *owner identifier*.

NOTE — If two public texts with the same owner have the same public text description, they must be of different classes, versions, etc.

10.2.2.1 Public Text Class

[86] public text class = *name*

The *name* must be one that identifies an SGML construct in the following list:

Name	SGML Construct
CAPACITY	<i>capacity set</i>
CHARSET	<i>character data</i>
DOCUMENT	<i>SGML document</i>
DTD	<i>document type declaration subset</i>
ELEMENTS	<i>element set</i>
ENTITIES	<i>entity set</i>
LPD	<i>link type declaration subset</i>
NONSGML	<i>non-SGML data entity</i>
NOTATION	<i>character data</i>
SHORTREF	<i>short reference set</i>
SUBDOC	<i>SGML subdocument entity</i>
SYNTAX	<i>concrete syntax</i>
TEXT	<i>SGML text entity</i>

The *name* must be entered with upper-case letters.

NOTE — When appropriate, a system can use the *public text class* to determine strategies for converting the public text from its interchange form into a referenceable entity that uses the system character set and concrete syntax.

10.2.2.2 Public Text Description

[87] public text description = *ISO text description*
| *minimum data*

An *ISO text description* can be used only when the *public identifier* identifies an ISO publication. It consists of the last element of the publication title, without the part number designation (if any).

NOTE — For example, the *ISO text description* for ISO 8632/4 is "Clear text encoding".

10.2.2.3 Public Text Language

[88] public text language = *name*

The *public text language* must be a two-character *name*, entered with upper-case letters. The *name* must be the language code from ISO 639 that identifies the principal natural language used in the public text.

NOTES

1 The natural language will affect the usability of some public text classes more than others.

2 The portions of text most likely to be influenced by a natural language include the data, defined names, and comments.

3 A system can use the *public text language* to facilitate automatic language translation.

10.2.2.4 Public Text Designating Sequence

[89] public text designating sequence
= *minimum literal*

The *minimum literal* must be the external form of a designating escape sequence prescribed by ISO 2022 for the character set referenced by the *public identifier*. If the public text is an ISO registered character set, the designating escape sequence must be a registered escape sequence for that set.

NOTES

1 For example, the external form of the registered G0 designating sequence for the graphic characters of ISO 646 IRV (registered character set 002) is:

ESC 2/8 4/0

2 For registered character sets, the *public text designating sequence* uniquely identifies the public text. For other character sets, it uniquely identifies the public text with respect to a particular *owner identifier*.

10.2.2.5 Public Text Display Version

[90] public text display version = *minimum data*

The *public text display version* must be omitted if the *public text class* is "CAPACITY", "CHARSET", "NOTATION", or "SYNTAX". For other classes, if the public text is device-dependent, the text identifier must include a *public text display version* that describes the devices supported or coding scheme used.

When a system accesses public text for which a *public text display version* could have been specified but was not, it must substitute the best available device-dependent version for the display device in use. If there is none, no substitution occurs.

NOTE — This mechanism is particularly useful with character entity sets.

10.3 Comment Declaration

[91] comment declaration = *mdo*, (*comment*, (*s* | *comment*)*)?, *mdc*

[92] comment = *com*, SGML character*, *com*

No markup is recognized in a *comment*, other than the *com* delimiter that terminates it.

10.4 Marked Section Declaration

[93] marked section declaration =
marked section start,
status keyword specification, *dso*,
marked section, *marked section end*

[94] marked section start = *mdo*, *dso*

[95] marked section end = *msc*, *mdc*

[96] marked section = SGML character*

The *marked section* must comply with the syntactic and semantic requirements that govern the context in which the *marked section declaration* occurs.

A *marked section end* that occurs outside of a *marked section declaration* is an error.

10.4.1 Quantities

The number of open marked section declarations cannot exceed the "TAGLVL" quantity.

10.4.2 Status Keyword Specification

[97] status keyword specification = (*ps* +,
qualified status keyword | *status keyword* |
"TEMP")*, *ps**

[98] qualified status keyword =
status keyword qualifier, *status keyword*

[99] status keyword qualifier = *name group*

[100] status keyword = "CDATA" | "IGNORE" |
"INCLUDE" | "RCDATA"

where

IGNORE specifies that the section is treated as though there were no characters in the *marked section*, except that a nested *marked section declaration* is recognized so that the correct ending can be found, but its *status keyword specification* is ignored.

INCLUDE specifies that the *marked section* is not to be ignored.

CDATA specifies that the *marked section* is treated as *character data*.

RCDATA specifies that the *marked section* is treated as *replaceable character data*.

TEMP identifies the section as a temporary part of the document that might need to be removed at a later time.

In the event of a conflicting specification, the status keywords have the following priority (highest shown first):

"IGNORE"
"CDATA"
"RCDATA"
"INCLUDE"

If none is specified, "INCLUDE" is assumed.

A *qualified status keyword* is ignored unless the *name group* includes the name of an active document type or active link type.

A *qualified status keyword* can be specified only if "CONCUR YES" or "EXPLICIT YES" is specified on the SGML declaration.

If the effective status is "CDATA" or "RCDATA", a nested *marked section declaration* within the *marked section* in the same entity is not allowed; the *marked section declaration* is terminated by the first *marked section end*.

If the effective status is "IGNORE", an **Ee** is not allowed in the *marked section*.

NOTE — The scan of an "IGNORE", "CDATA", or "RCDATA" marked section ignores virtually all markup but marked section ends. As a result, processing instructions, attribute values, literals, character data elements, and comments are not recognized to be such, so their characters are also scanned. This could cause erroneous results if the characters look like marked sections. In most cases, problems can be avoided by entering such characters with references, instead of directly.

10.5 Entity Declaration

[101] entity declaration = *mdo*, "ENTITY", *ps* + ,
entity name, *ps* + , entity text, *ps**, *mdc*

10.5.1 Entity Name

[102] entity name = *general entity name* |
parameter entity name

[103] general entity name = *name* | (*rni*,
"DEFAULT")

[104] parameter entity name = *pero*, *ps* + , *name*

where

DEFAULT means the entity is the default entity.

An attempt to redefine an entity is ignored, but is not an error.

NOTE — This requirement allows an entity declaration in a document type declaration subset to take priority over a later declaration of the same entity in a public document type definition.

The *ps* in a *parameter entity name* is required, even though it follows a delimiter.

10.5.1.1 Quantities

A *name* in a *parameter entity name* must be at least one character shorter than the "NAMELEN" quantity.

10.5.1.2 Capacities

Points are counted towards the "ENTCAP" capacity for the default entity, and for each unique entity name that is defaulted in one or more references.

Points are counted towards the "ENTCHCAP" capacity for the default entity. They are counted for

defaulted entities only if a system identifier is generated.

10.5.2 Entity Text

[105] entity text = *parameter literal* | *data text* |
bracketed text |
external entity specification

If a *parameter literal* alone is specified as the *entity text*, the interpreted *parameter literal* is the replacement text of the entity.

An **Ee** is deemed to be present at the end of the replacement text; it is not entered explicitly.

10.5.3 Data Text

Data text is treated as *character data* when referenced, regardless of the context in which the entity reference occurs. It is specified as a *parameter literal*, whose characters, after resolution of references in the usual manner, will comprise the *entity text*.

[106] data text = ("CDATA" | "SDATA" | "PI"),
ps + , *parameter literal*

where

CDATA means the interpreted *parameter literal* is the replacement text of a character data entity.

SDATA means the interpreted *parameter literal* is the replacement text of a specific character data entity.

PI means the interpreted *parameter literal* is the replacement text of a processing instruction data entity.

"CDATA" or "SDATA" can be specified only if the *entity name* is a *general entity name*.

A *processing instruction* that returns data must be defined as an "SDATA" entity.

NOTES

1 A "CDATA" or "SDATA" entity must be referenced in a context in which *character data* can occur, and a "PI" entity in a context in which a *processing instruction* could occur.

2 "SDATA" is normally specified if the entity would be redefined for different applications, systems, or output devices; for example, if the data contained processing instructions, or characters not present in the *translation-reference character set*.

10.5.4 Bracketed Text

[107] bracketed text = ("STARTTAG" | "ENDTAG"
| "MS" | "MD"), *ps* + , *parameter literal*

where the keywords mean that the entity consists of the interpreted *parameter literal* bracketed with delimiters, as follows:

STARTTAG	means preceded by stago and followed by tagc .
ENDTAG	means preceded by etago and followed by tagc .
MS	means preceded by a <i>marked section start</i> and followed by a <i>marked section end</i> .
MD	means preceded by mdo and followed by mdc .

NOTE — Bracketed text is simply text with delimiter characters; there is no requirement that the entities form valid start-tags or other constructs. As usual, the validity of the entity is determined in context whenever the entity is referenced.

10.5.4.1 Quantities

The length of an interpreted *parameter literal* in *bracketed text* cannot exceed the "LITLEN" quantity, less the length of the bracketing delimiters.

10.5.5 External Entity Specification

[108] external entity specification
= *external identifier*, (*ps* + , *entity type*)?

[109] *entity type* = "SUBDOC" | ("NDATA", *ps* + ,
notation name)

where

SUBDOC	means the entity is an SGML <i>subdocument entity</i> .
NDATA	means the entity is a <i>non-SGML data entity</i> .

The *entity type* can be specified only if the *entity name* is a *general entity name*.

If the *entity type* is omitted, the entity is an SGML *text entity*.

The *notation name* must be declared in the same document type definition as the entity.

NOTE — It need not have been defined prior to this declaration, but must be defined prior to a reference to the entity.

A non-SGML data entity can reference (in its own notation) other non-SGML data entities and SGML sub-document entities. Such entities should be declared in the same document type definition as it is.

"SUBDOC" can be specified only if "SUBDOC YES" is specified on the SGML *declaration*.

11 Markup Declarations: Document Type Definition

11.1 Document Type Declaration

[110] document type declaration = **mdo**,
"DOCTYPE", *ps* + , *document type name*,
(*ps* + , *external identifier*)?, (*ps* + , **dso**,
document type declaration subset, **dsc**)?,
*ps**, **mdc**

[111] document type name = *generic identifier*

[112] document type declaration subset =
(*entity set* | *element set* |
short reference set)*

[113] *entity set* = (*entity declaration* | *ds*)*

[114] *element set* = (*element declaration* |
attribute definition list declaration |
notation declaration | *ds*)*

[115] *short reference set* = (*entity declaration* |
short reference mapping declaration |
short reference use declaration | *ds*)*

The *document type name* must be a *generic identifier* that does not occur as a *document type name* or *link type name* in the same *prolog*.

The *external identifier* points to an entity that is referenced at the end of the declaration subset and is considered to be part (or all) of it. The effective document type definition is the combination of the declarations entered in the subset and the external ones.

NOTE — A parameter entity declaration in the subset will have priority over another declaration for that entity in the external entity, as the external entity is parsed later.

A *document type declaration* must contain an *element declaration* for the *document type name*.

A *short reference set* is permitted only in the base document type declaration.

11.1 Document Type Declaration