IBM ID Doc
Language Reference
Draft 1.4

Document Number SGMLBOOK-REF-0

February 13, 1994

IBM Software Doc Design Team
Wayne Wohler
Don R. Day
W. Eliot Kimber
Simcha Gralla
Mike Temple

&@aline1(*).

Notice to reader:

This document is the first public draft of this material and as such is subject to revision. This document is not a commitment by IBM to provide any function or feature as a product or service at any time in the future.

Contents

About This Information ix
Applicable Standards ix
What else to read x

Forward xi
A Brief History of IBM ID Doc xi
Invocation xi

Part 1. Introduction to IBM ID Doc 1

Chapter 1. Understanding How IBM ID Doc Relates to DCF BookMaster . 2
Basic Terms and Concepts 2
The Difference Between Elements and Tags 2
Containment 4
Specifying Elements in the Right Order 5
DCF Symbols and SGML Entities 6
Conditional Processing with IBM ID Doc 7
Common Attributes 8

Chapter 2. IBM ID Doc Hyperlinking 10
Implicit or Associative Linking 10
Direct or Explicit Links 12
Cross Reference Links 12
Content Reference Links 12
Retrieval Alternatives 13
Property Reference Links 13
Generic Hyperlink 14
Linking to Other Documents 14
Using HyTime Location Elements with Explicit Links 14
Qualified Element IDs 16
Property Inheritance 18
Using The PropertyDef Element 18
Using The ClassDef Element 19

Chapter 3. Extending The IBMIDDoc Document Type Locally 22

Part 2. Language Reference 23

Chapter 4. Introduction 24
Overview of the Reference Entries 24
ElementName Short Description 24
How to Read the Syntax Diagrams 24
How to Read the SGML Definitions 25
Element Declarations 26
Attribute List Declarations 27
How to Read HyLex Specifications 28

Chapter 5. The IBMIDArch Architecture 30

© Copyright IBM Corp. 1992, 1993

Chapter 6. IBM ID Doc Modules 32
Module and Function Use Declaration 32
Organization of IBM ID Doc Modules 32
IBM ID Doc Base Element Set 34

Chapter 7. IBM ID Doc Elements 35
Abbreviations 38
Abstract 39
Address 40
AnchorRoleDef (hyperlink anchor role defininition) 45
Annot (annotation) 46
Annotation Anchor Role (annotation link) 48
APL (APL data) 49
Appendix 50
Author 51
Authors 52
Backmatter 53
Bibliography 55
BibList (bibliography entry list) 56
BibRecord (bibliographic record) 58
Bin (binary or Boolean data) 62
Body (document body) 63
Bridge (bridge between concepts) 64
Caption 67
Caution (caution notice) 67
CGraphic (character graphic) 69
Char (character data) 70
Citation (document citation) 71
Class Attribute (element class) 73
ClassDef (element class definition) 74
Common Attributes 77
ControlInfo (process-related control information) 81
CopyRight (copyright attribution) 85
CopyRights (copyright attributions) 88
Danger (danger notice) 89
Date 90
Decimal (decimal number) 91
DecisionPoint 92
DefinedTerm 93
Delim (syntactic delimiter) 94
Desc (element's description) 95
DescribedDocument Anchor Role (link to a described document) 96
Div (hierarchical division) 97
DivBody (division body) 99
DivIntro (division introduction) 100
DivProlog (division metainformation) 104
DivSummary (division summary) 106
DL (definition list) 107
DLBlock (definition list block) 108
DLEntry (definition list entry) 109
DocTitle (document title) 111
DocTrackingInfo (document tracking and control information) 113
EditionNotices 115
Enterprise (enterprise name and address) 116

EnterpriseName (enterprise name) . . . 117
FigSeg (figure segment) . . . 118
Figure (exhibits) . . . 119
FigureList (list of figures) . . . 121
FN (footnote) . . . 122
Fragment (syntax fragment) . . . 124
FragmentRef (syntax fragment reference) . . . 125
Frontmatter . . . 127
GL (glossary list) . . . 129
GLBlock (glossary list block) . . . 131
GLEntry (glossary list entry) . . . 132
Glossary . . . 134
Group (syntax grouping) . . . 135
Hex (hexadecimal number) . . . 138
Hyperlink Attributes (hyperlink definitions) . . . 140
IBMBibRecord (IBM bibliographic record) . . . 141
IBMIDDOC (IBM-specific product documentation) . . . 145
IBMLibRecord (IBM document library definition) . . . 147
IBMSafety (IBM safety notices) . . . 150
ID Attribute (element identifier) . . . 152
Index . . . 152
InfoModule (information module) . . . 154
IRef (index entry reference) . . . 156
I1 (first-level index entry) . . . 157
I2 (second-level index entry) . . . 159
I3 (third-level index entry) . . . 161
Kwd (syntactic keyword) . . . 163
LE (language element) . . . 164
LEDesc (language element description) . . . 165
LEDI (language element description item) . . . 166
Legend . . . 169
LEName (language element name) . . . 171
LERS (language element reference section) . . . 172
LERSDef (LERS property definition) . . . 173
LI (list item) . . . 175
LIBlock (list item block) . . . 176
LibRecord (document library definition) . . . 177
Lines (text with significant line boundaries) . . . 181
Link (explicit hyperlink) . . . 182
LinkDesc (linked location description) . . . 184
LinkEnds, AnchorRoles, and HyTime Attributes (hyperlink target reference attributes) . . . 187
Litdata (literal data) . . . 190
LongQuote (quotation that stands alone) . . . 192
ManPage (Unix manual page) . . . 193
Mark (marked note definition) . . . 195
MarkAction (marked note action definition) . . . 196
MarkClass (marked note class definition) . . . 197
MarkDesc (mark description) . . . 199
MarkList (marked note list) . . . 200
MarkNote (marked note) . . . 202
MD (deleted phrase) . . . 204
Message (message or code description) . . . 205
MessageItem (message description item) . . . 206

MessageList List of message descriptions . . . 209
ModularInfoDef (modular information property definition) . . . 210
ModularInformation . . . 212
ModuleItem (module description item) . . . 213
MultiMediaObject (non-text object) . . . 215
MV (message variable) . . . 217
Name (person's name) . . . 218
Nameloc (named location) . . . 220
NMList (list of IDs or entities) . . . 222
NMQuery (named query) . . . 225
Notation Attributes . . . 227
Note . . . 230
NoteList (ordered note list) . . . 231
Notloc (notation-specific location) . . . 232
Number . . . 233
ObjectContainer (data object container) . . . 234
ObjectLibrary . . . 236
Octal (octal number) . . . 238
OL (ordered list) . . . 239
Oper (syntactic operator) . . . 240
P (paragraph) . . . 241
ParaBlock (paragraph block) . . . 243
ParaLevelElements (paragraph-level elements) . . . 245
Parm (parameter list entry) . . . 247
ParmBlock (parameter list block) . . . 248
ParmList (parameter list) . . . 249
Part (major document part) . . . 251
Person (person's name and address) . . . 253
Phrase (one or more words) . . . 254
PhraseLevelElements . . . 256
Preface . . . 257
Procedure . . . 258
ProcEntry (procedure entry point) . . . 260
ProcExit (procedure exit point) . . . 262
ProcStep (procedure step) . . . 263
Prolog (document metainformation) . . . 266
Property Attributes (element properties) . . . 268
PropertyDef (property set definition) . . . 271
PropertyDefinitions . . . 272
PropertyGroup . . . 273
PropertySource Anchor Role (property reference link) . . . 275
PublicIdentifier . . . 276
Quote (quotation phrase) . . . 276
Range (syntactic range) . . . 278
RefKey (reference key hyperlink) . . . 281
RepSep (repeat separator) . . . 284
Alternate Anchor Role (retrieval alternative link) . . . 286
Revision . . . 287
RevisionHistory (revision tracking information) . . . 288
RevisionSpecification Anchor Role (revision specification link) . . . 289
Safety (safety notices) . . . 290
Screen (computer screen or panel) . . . 291
Set (syntactic set) . . . 293
SOA (summary of amendments) . . . 296

StepNotes . . . 297
Style Attribute (element style information or reference) . . . 298
Syntax . . . 301
SyntaxBlock . . . 304
SyntaxNote . . . 306
SyntaxPhrase (syntax phrase) . . . 307
Table . . . 308
TableList . . . 311
Title (element title) . . . 312
TOC (table of contents) . . . 314
UL (unordered list) . . . 316
Var (syntactic variable) . . . 317
Warning (warning notice) . . . 318
Web (hyperlink web) . . . 319
WebNode (hyperlink web node) . . . 321
WebSet (set of hyperlink webs) . . . 324
Xmp (example of computer input or output) . . . 325
XPH (example phrase) . . . 326
XRef (cross reference) . . . 327

Part 3. Appendixes . . . 331

Appendix A. Proposed IBM Standard for Formal Public Identifiers . . . 332

Appendix B. IBM ID Doc Naming Conventions . . . 338

Appendix C. Predefined HyQ Query Functions for IBM ID Doc . . . 339
All_BibRecords() . . . 339
All_Divisions_To_Level(level) . . . 340
All_Figures() . . . 340
All_Figures_of_Class_X() . . . 341
All_GLEntries() . . . 342
All_LibRecords() . . . 342
All_Safety_Notices() . . . 343
All_Tables() . . . 343

Appendix D. IBM ID Doc Document Type Definition . . . 345

Glossary . . . 412

Notices

Reference in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only a program or product may be used.

Any functionally equivalent product which does not infringe any IBM intellectual property rights may be used instead of the IB product. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBI user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this docum does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

| A HyTime application conforming to International Standard ISO/IEC 10744-- Hypermedia/Time-based Structuring Language.

About This Information

This information documents the IBM Software Document (IBMIDDOC) SGML application. The application definition is defined by a combination of SGML element specifications (the IBMIDDOC DTD) and associated application-specific semantics. This definition does not imply or require a particular implementation. It represents an application design divorced from any specific implementation. The design has been constructed so as to provide enough optional features and scalability in order to allow multiple different implementations of differing complexity, completeness, and cost.

The IBMIDDOC application is based on the IBM InfoMaster Architecture, which defines a set of architectural forms for technical documents and set of base processing semantics for those forms. The InfoMaster Architecture is documented separately. It is not necessary to understand the InfoMaster Architecture in order to understand this design.

In addition, the IBMIDDOC language represents a very rich language designed to satisfy the onerous requirements of IBM's product documentation. It is expected that for specific applications, subset languages will be defined that are a proper subset of the IBMIDDOC language. For example, it would be possible to define an application that uses only the modular information elements of IBMIDDOC, for example, in an online help application. The InfoMaster Architecture and IBMIDDOC are intended to provide a high degree of flexibility while enabling information interchange and re-use to the greatest degree possible.

This design is intended to be a general design, implementable on all platforms using a variety of SGML parsers and composition and presentation systems. IBM ID Doc has been specifically designed to meet a wide range of print and non-print text-presentation applications, including the use of IBMIDDOC source files as databases used for information retrieval.

This document is the the definitive definition of the IBM ID Doc application.

The complete information on how to use IBM ID Doc as an author is in *The IBM ID Doc User's Guide*.

Applicable Standards

IBM ID Doc conforms to the following standards:

- ISO 8879 Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)
- ISO/IEC 10744 Information Processing — Hypermedia Time-based Systems (HyTime).
- ISO 9070, Information processing — SGML support facilities — Registration procedures for public text.
- ISO/TR 9573: Information Processing — SGML Support Facilities — Techniques for Using SGML.

In addition, IBM ID Doc conforms to the IBM InfoMaster Architecture. See *InfoMaster Architectural Form Reference*.

© Copyright IBM Corp. 1992, 1993

What else to read

Related publications you might also want to refer to are:

BookMaster

- *IBM BookMaster General Information*, GC34-5006
- *IBM BookMaster User's Guide*, SC34-5009
- *IBM BookMaster Creating Named Styles*, SC34-5008
- *IBM BookMaster Text Programmer's Notebook*, SC34-5012

SGML Translator

- *IBM Publishing Systems SGML Translator General Information*, GC34-5071
- *IBM Publishing Systems SGML Translator Creating a Document for DCF Processing*, SC34-5076
- *IBM Publishing Systems SGML Translator Programming Guide and Reference*, SC34-5072
- *IBM Publishing Systems SGML Translator Creating a DTD and an SGML Declaration*, SC34-5075
- *IBM Publishing Systems SGML Translator Validating and Translating*, SC34-5074

Forward

The clichéd metaphor “on the shoulders of giants” could not be more true in the case of IBM ID Doc. The core IBM ID Doc design team is:

Wayne Wohler, Team Leader
W. Eliot “Dr. Macro” Kimber
Don Day
Simcha Gralla

We owe a tremendous amount to Truly Donovan, the chief designer of BookMaster, and before it, ISIL; and to the OSF DocSig, out of whose work we developed many of our ideas and from whom we learned a lot about how things are and the way people think outside of IBM.

We have made an effort to involve the BookMaster user community as we have developed the design. The IBM ID Doc language is really owned by the people who use it. We depend on our users to help us decide what the best thing to do is.

IBM ID Doc was also heavily influenced by the HyTime standard and the path it charted for the rigorous definition of document architectures.

A Brief History of IBM ID Doc

IBM ID Doc is a direct descendant of ISIL and DCF BookMaster, representing an evolutionary development. In many ways, IBM ID Doc represents a significant change in direction from BookMaster. The use of SGML coupled with HyTime and the power of online presentation and hypertext led us to largely re-invent the philosophical underpinnings of BookMaster. As result, many concepts embodied in DCF BookMaster have been refined and generalized, and several concepts are new with IBM ID Doc, most notably the concept of annotations as a separate class of data distinct from document content and metadata, and the concept of generic containment and use-by-reference of element content. IBM ID Doc broadens its focus beyond its print origins and refocuses on online presentation and retrieval. We have removed all print-specific constructs and dependencies while adding base concepts and constructs meaningful only in an online, dynamic, hyperlinked universe.

Invocation

BookMaster is about words. It has been, over its life, developed by people who live in the world of words for use by word people. It therefore seems appropriate to include in this document an invocation to the muse of writing to bless this design and those who use it. For this invocation we have chosen a poem by William Butler Yeats. By pure accident, this poem happens to be particularly significant to BookMaster because of BookMaster's progenitor, ISIL,[1] the IBM internal GML language that became BookMaster. In the fictional mythology of J.R.R. Tolkien, author of *The Lord of the Rings* and *The Hobbit*, Isil is the moon. In the poem that

[1] ISIL stands for “information structure identification language”.

© Copyright IBM Corp. 1992, 1993

follows, the moon is induced to change by Minnaloushe, the cat; at the same time, Minnaloushe learns from and is moderated by the moon. Similarly, IBM ID Doc has induced ISIL to change, bringing it to a new phase; at the same time, ISIL still informs and moderates the design of IBM ID Doc.

The Cat and the Moon[2]

The cat went here and there
And the moon spun round like a top,
And the nearest kin of the moon,
The creeping cat, looked up.
Black Minnaloushe stared at the moon,
For, wander and wail as he would,
The pure cold light in the sky
Troubled his animal blood.
Minnaloushe runs in the grass
Lifting his delicate feet.
Do you dance, Minnaloushe, do you dance?
When two close kindred meet,
What better than call a dance?
Maybe the moon may learn,
Tired of that courtly fashion,
A new dance turn.
Minnaloushe creeps through the grass
From moonlit place to place,
The sacred moon overhead
Has taken a new phase.
Does Minnaloushe know that his pupils
Will pass from change to change,
And that from round to crescent,
From crescent to round they range?
Minnaloushe creeps through the grass
Alone, important and wise,
And lifts to the changing moon
His changing eyes.

—William Butler Yeats, *The Wild Swans at Coole* (1919)

[2] You can make this a multimedia document by reading this poem aloud.

Part 1. Introduction to IBM ID Doc

Chapter 1. Understanding How IBM ID Doc Relates to DCF BookMaster . 2

Chapter 2. IBM ID Doc Hyperlinking 10

Chapter 3. Extending The IBMIDDoc Document Type Locally 22

© Copyright IBM Corp. 1992, 1993

Chapter 1. Understanding How IBM ID Doc Relates to DCF BookMaster

Note: This section is intended for current users of DCF BookMaster who need to understand IBM ID Doc. It is assumed you already have a working knowledge of DCF BookMaster.

IBM ID Doc represents an evolution of the BookMaster language. In the course of this evolution, we have introduced some new concepts and terms. None of this new stuff is particularly complex, but is just enough different from what you may be familiar with to be confusing. The following relates what you already know about DCF BookMaster to the new concepts and terms in IBM ID Doc.

IBM ID Doc is *SGML*; this means *Standard Generalized Markup Language*, which makes it a standardized language, and more open than DCF BookMaster. More products are being developed all the time that accept SGML documents, including WYSIWYG editors and formatters.

Basic Terms and Concepts

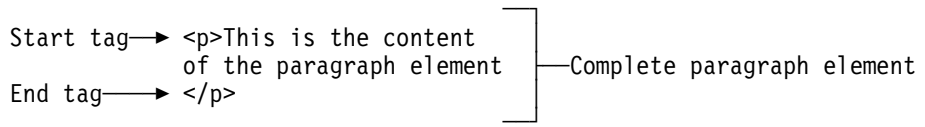
It is not necessary to be an SGML guru in order to use IBM ID Doc effectively. However, there are a few basic SGML concepts and a few specialized terms that you will need to know in order to make the best use of IBM ID Doc. These terms and concepts are formalisms of aspects of GML and BookMaster that you are probably already familiar with. The terminology that SGML defines is very precise and all SGML terms have a very precise meaning. Every effort has been made in the IBM ID Doc documentation to use the SGML terms as they are intended to be used and as they are defined in the SGML standard, ISO 8879. One of the values of standardization is that once you learn SGML, for any application, that knowledge is transferable to any other SGML application. As for any new subject, you may find the SGML jargon somewhat stilted and confusing at first, but SGML is very logical and once you get the hang of it it will all make sense.

There are many aspects of SGML that as an author of SGML documents you need never be aware of. There is absolutely no reason, for example, for you to actually read the SGML standard. The IBM ID Doc documentation contains all the SGML details you need, and hopefully only those aspects of SGML you really need to understand.

A good analogy to SGML are SQL databases. SQL is very complicated, but you only need to know a little bit about SQL in order to use SQL applications. So it is with SGML.

The Difference Between Elements and Tags

In SGML, a distinction is made between *elements* and *tags*. An element represents a complete thing, say a paragraph, including any elements that might be within the paragraph. A tag is just the marker that delimits the start and end of the element.



© Copyright IBM Corp. 1992, 1993

Notice that the tags begin and end with different characters than you are used to in DCF BookMaster. `<tag>` indicates the start, and `</tag>` indicates the end.

In SGML languages like IBM ID Doc, some start and end tags can be omitted. IBM ID Doc has been designed so that you don't have to enter tags you wouldn't normally expect to from using DCF BookMaster. Omission is important because it makes it easier to create SGML documents when you are using a plain text editor (as opposed to a specialized SGML editor). Omission is also important because it is important to understand that tags you may not normally type and that you may not normally think about are implied. For example, you may occasionally get an error message that refers to an element whose tags you didn't type. That element has an omissible tags that were implied automatically by the SGML parser. Such an error message would be impenetrably confusing if you didn't understand how tag omission and implied elements work.

The division element in IBM ID Doc demonstrates how omitted tags work.

In IBM ID Doc, all the headings (H0 to H20 in DCF BookMaster) have been replaced by a single element called Div, for “hierarchical division.” The term *division* is what DCF BookMaster calls a *heading*. The difference is that the term division refers to everything from one “heading” at the same level to the next. In IBM ID Doc, the heading is the division title.[3]

Div contains two elements, DivProlog and DivBody. DivProlog then contains the heading title. DivBody then contains the content of the division, including any nested Div elements. This sounds very complex, but in actual practice, it looks just like it did in DCF BookMaster:

```
<div>Division Title
<p>Division content.
</div>
```

In this example, the DivProlog, Title, and DivBody tags are automatically implied. This implication is a function of SGML that IBM ID Doc takes advantage of. Note that, superficially, the example above looks almost exactly like a normal DCF BookMaster heading. However, under the covers, the markup looks like this:

```
<div><divprolog><title>Division Title</title></divprolog>
<divbody>
<p>Division content.
</divbody>
</div>
```

As you can tell from these examples, elements exist whether or not you see the markup.[4] The SGML parser knows that the title text is contained by the Title element, regardless of whether or not you actually type in a Title tag. As a rule,

---

[3] Specifically, the TOSL text of heading tags in DCF BookMaster is contained in IBM ID Doc by the Title element.

[4] This function is enabled by the SGML *document type definition*, or DTD. The DTD uses a formal language similar to BNF expressions to define the precise rules for all elements, including:

- What other elements a given element can contain
- Whether or not those elements are optional or required
- Whether or not those elements can be repeated
- The attributes allowed on an element
- Whether or not an attribute is optional or required and what the default value is, if any.

you can omit the start tag for any element that is required. In the case of the Div element, the title is required, allowing you to omit the Title start tag. For elements that have an optional title, you must explicitly enter the Title start and end tags. For example, a syntax diagram has an optional title, so you must code the Title tag explicitly.

However, the short title element, contained by Title, is optional. Thus, when you want to include a short title, you must enter the ShortTitle tag:

```
<div><ShortTitle>Short title</>
This is the long title
<p>Division content.
  ⋮
</div>
```

Note the </> following the short title. This is called an *empty end tag* in SGML. It is a shorthand way of ending the last element started, in this case, the ShortTitle element.

Note about SGML Editors

The discussion of entering tags is in the context of using a plain text editor like Xedit or EPM to create SGML files. In this context, minimizing typing is useful, and IBM ID Doc does what it can to keep typing to a minimum.

However, you can also create and edit SGML files with editors that provide direct support for SGML constructs in a more visual or WYSIWYG way. These editors are much like traditional word processing programs like Word for Windows or Ami Pro where you select “styles” from menus rather than typing in tag names. Because you never have to actually type tags in these editors, tag minimization is not an issue for them. When you use an SGML editor and you request a view that shows the markup or you create the actual SGML file, you may see tags that you normally wouldn't have to type.

Containment

Another important SGML concept is *containment*. DCF BookMaster also has the concept of containment, but in IBM ID Doc containment is more important because SGML defines containment completely and unambiguously in a way that DCF BookMaster does not.

In any SGML language, elements contain things, and all things are contained within elements. The TOSL feature of DCF BookMaster does not exist in IBM ID Doc. In DCF BookMaster, the text on the same line as a tag (the TOSL text) has special meaning and is processed differently from the text following it on subsequent lines. This meant, for example, that you couldn't put tags in the text of headings. This feature does not exist in SGML and thus is not a part of IBM ID Doc. Instead, elements like Title are used to contain title text. This means, for example, that heading text can now contain other elements and be entered on many lines.

The DTD is a key aspect of SGML that distinguishes it from DCF GML. The DTD makes it possible for any SGML parser to validate any SGML document without knowing anything about how the document should be processed to create output.

Containment implies a start and an end. In SGML, all elements have a start and an end. For some elements, the end tag is optional, for others it is required. For example, the paragraph element (P) has an optional end tag because a paragraph is automatically ended by the beginning of another paragraph. However, the Div element must be explicitly ended because Div can itself contain nested Divs. It is never wrong to explicitly end an element, except in a few special cases.

There are a few elements that don't have end tags. These are the elements that don't contain anything, specifically, the XRef element in IBM ID Doc. Also, all elements let you choose whether or not to have the content be inline or used by reference. The LitData element is an example of this behavior. In IBM ID Doc, as in DCF BookMaster, LitData can either contain literal data in line or can refer to an external file to be embedded. When you refer to an external file, it is an error to code a LitData end tag.

Another aspect of containment is *direct containment*. In the example below, the paragraph directly contains the ordered list, the ordered list directly contains the list items, and the first list item directly contains a paragraph within itself:

```
<p>This is a list:
<ol>
<li>List item one
<p>Paragraph within list item one.
<li>List item two
</ol>
```

The distinction between general containment and direct containment is important because some elements cannot directly contain other elements, but may be able to contain other elements that do contain them. For example, paragraph cannot directly contain another paragraph, but it can contain elements that themselves contain paragraph, as in the example above.

In IBM ID Doc, direct containment is used to determine implicit associations between elements. For example, a footnote is implicitly associated (linked) to the element that directly contains it. Thus, a footnote that is immediately contained by a Div is implicitly linked to the division as a whole, while a footnote within a paragraph is only implicitly linked to the paragraph, and not to the Div that contains the paragraph.

Also, the containment structure is used to determine the inheritance of properties from an element to its children. See “Property Inheritance” on page 18 for more information on property inheritance with IBM ID Doc.

Specifying Elements in the Right Order

In DCF BookMaster, many elements that could only be specified once could be specified in any order. For example, in the Prolog element, you can only code one DocNum, but you can code it anywhere. In IBM ID Doc, this freedom is no longer available. When elements can only occur once, they must be coded in the order shown in the syntax definitions and the formal SGML definitions. This restriction is imposed by the intricacies of SGML which make “code once but in any order” constructs problematic in some situations.

Note that this requirement to specify things in a certain order should not be a problem when you are using an SGML editor because the editor will always know the right order and should not let you specify things incorrectly.

DCF Symbols and SGML Entities

In DCF you have symbols and embedded files. DCF symbols are dynamic in that you can change a symbol's value at any point in a document. SGML does not have dynamic symbols. Instead it has *entities*.

In SGML, an entity is simply a document fragment, anything from a single character to an entire document. Entities are not dynamic. Once an entity has been declared and defined, it cannot be changed. In addition, all entities must be defined at the beginning of a document.

SGML defines two basic types of entities, internal entities and external entities. Internal entities are more or less the same as BookMaster .NAMEIT symbols, relatively short bits of text defined within your document. The main differences between .NAMEIT symbols and internal entities is that internal entities are not dynamic, meaning you can't change their value within a document, and they can contain markup.

Internal Entities

What DCF calls a symbol is called an *entity reference* in SGML. For example, say you define an entity called 'prodname' that contains the name of your product. You use that entity by coding an entity reference, like so:

```
The &prodname; system is better than sliced bread.
```

Note that in SGML an entity reference ends with a semicolon and not a period.

Like DCF, entity names are case sensitive in IBM ID Doc. In just about all ways, entity references to internal entities behave just as symbols do in DCF. Internal entities are defined like so in SGML, using an *entity declaration*:

```
<!ENTITY prodname  "Macro-O-Matic" >
```

In SGML all entities must be defined at the start of the document[5] and cannot be redefined within a document.

External Entities

External entities are files such as you would embed in DCF BookMaster with the .IM macro. However, in SGML, instead of using .IM to embed a file, you simply code the entity reference for the file, just like you would code a symbol in DCF BookMaster. For example, to embed a file into your document, you first define the entity using a slightly different form of

```
<!ENTITY aboutmyprod-usage  SYSTEM "ABCL0USE SGML"  -- Usage chapter -->
```

The rules for entity declarations are defined by the SGML standard and are the same for all SGML applications. The external entity declaration is similar in function to the DCF BookMaster .namefile macro. As with .namefile, the entity declaration associates a local symbolic name (aboutmyprod-usage in this example) with an actual system-specific filename (ABCL0USE SGML in this example).

To embed the file, you simply code an entity reference, like so:

---

[5] Technically, they are part of the DTD itself and are normally put in what is called the DTD subset, which is that part of the DTD that is specific to a given document, as opposed to the part of the DTD that authors never see or modify. Most SGML editors will handle the details of defining entities for you.

```
<!-- Embed the usage chapter: -->
&aboutmyprod-usage;
```

Special Characters

IBM ID Doc uses the standard symbol sets defined by International Standards Organization (ISO), as well as a few symbols that DCF BookMaster defines that the ISO does not. The symbol names are, for the most part, different from those defined by DCF BookMaster, but the same special characters are available. This move to the ISO symbol sets makes IBM ID Doc more compliant with international standards than it would be if it continued to use the old symbol names.

Conditional Processing with IBM ID Doc

DCF BookMaster provides conditional processing using the following mechanisms:

- PSC elements
- ANNOT elements
- DVCF macros

All of these mechanisms require *sequential processing* to be properly resolved. In other words, the document must be processed from start to finish, doing all symbol resolution, resolving DVCF correctly, and so on, in order to know what is in and what is out. They also affect the validity of documents based on the conditions specified. In other words, a document may be valid when one set of conditions is specified and invalid when a different set is specified. It is impossible to look at a just a fragment of a document and know what the correct values for any symbols or DVCF conditions are.

The abstract processing model used for processing IBM ID Doc documents is *retrieval*, where the processor, the thing that actually creates something from the source files, retrieves elements as it needs them. By matching property values, the processing application can retrieve only those elements it needs. Conditional processing is much simpler for you as an author because as long as you code the correct attribute values, things will work correctly. There is no need to code complex DVCF conditions or clutter your markup with PSC or REV tags, which so easily get out of place.

IBM ID Doc has replaced all these methods of conditional processing with a simple but powerful method. All conditional processing is done by evaluating certain attribute values on elements to determine whether or not those elements should be processed. The attributes are called *property attributes* because they define properties of elements. A property is anything that describes an element, such as its owner, revision status, security classification, and so on. IBM ID Doc defines a number of property attributes that you can use to describe elements. When you process a document, you define the set of property values to be used to decide which elements to retrieve for processing.

The basic property attributes are:

PROC=

Defines the processes this element should be used for. This is analogous to the PROC= attribute on the DCF BookMaster PSC tag.

XPROC=
Defines the processes this element should **not** be used for.

REVISION=
Indicates the revision or revisions this element corresponds to.

STATUS=
Indicates the revision status, one of UCHANGED, NEW, CHANGED, or DELETED. Deleted elements can be automatically suppressed.

QUALIFICATION=
Refers to one or more qualification definitions, which are analogous to revision definitions.

LANGUAGE=
Indicates the language this element is written in. This enables the creation of multi-language documents.

SECURITY=
Security classification.

VERSION=
Defines the document version this element belongs in. Analogous to the DVCF .WHEN condition.

XVERSION=
Defines the document version this element does not belong in. Analogous to the DVCF .WHEN condition.

LEVEL=
Defines the document level this element belongs in. Analogous to the DVCF .WHEN condition.

XLEVEL=
Defines the document level this element does not belong in. Analogous to the DVCF .WHEN condition.

Common Attributes

Most of the attributes defined by IBM ID Doc are common to all elements. Attributes are divided into the following basic classes:

Identifying Attributes
This is the ID= attribute and any other attributes that may serve to identify a given element. All elements can take an ID= attribute.

Property Attributes
These are the property attributes, as described above.

Hyperlink Attributes
Attributes that define hyperlink relationships between elements. IBM ID Doc defines a rich set of hyperlink types. Any element can be the start of a hyperlink.

There are no presentation-definition attributes in IBM ID Doc as there are in DCF BookMaster, just the single Style= attribute. The Style= attribute in its pure form is intended to be used to define a link between a given element and a style definition for that element specified in a separate style definition. One of the key tenets of SGML is the complete separation of content and style. However, pragmatic considerations do not allow the use of such a pure system in all cases, so Style= can also

be used to contain application-specific presentation style control information. For the purposes of migrating existing DCF BookMaster documents into IBM ID Doc, DCF BookMaster presentation attributes can be specified as the content of the Style= attribute, like so:

```
<figure style="place=inline width=column frame=rules">
```

Note that this use of the Style= attribute is purely a pragmatic concession to current system limitations. It will eventually be replaced by a more integrated authoring and production system where the details of how style information and element content get associated is managed transparently to you the author while maintaining, under the covers, the complete separation of content and style.

Chapter 2. IBM ID Doc Hyperlinking

IBM ID Doc provides a rich set of hyperlinking functions and semantics. IBM ID Doc defines hyperlinking as connecting two or more elements together for whatever reason. All relationships between elements in IBM ID Doc information are defined in terms of hyperlinks, including the structural relationships defined by the structure of the SGML markup.

IBM ID Doc defines two mechanisms for defining hyperlinks: implication and direct reference. Links are defined by direct reference using element IDs and ID references. Cross reference and explicit hyperlink phrases are typical examples of direct reference links. IBM ID Doc defines a variety of direct reference semantics. Implicit hyperlinks are derived from the structure of the markup.

Implicit or Associative Linking

Links are defined by implication by associating hyperlink semantics with specific elements or structures. For example, a footnote is implicitly linked to the element that contains it. In IBM ID Doc the nesting of elements defines the hierarchical links between elements. For example, a paragraph nested within a division is implicitly hyperlinked to the division with a hierarchical link. IBM ID Doc defines a variety of implicit link semantics and specific applications or presentation systems are free to define new implicit link semantics.

Implicit hyperlinking is aided by the use of the common IBM ID Doc Class= attribute to associate author-defined classes with elements. For example, you could define a phrase class of sensecode and then defined implicit link semantics to phrases of that class. You could define a hyperlink style that associates any phrase of class sense sensecode with information modules of class sensecode that have the sensecode text in their name.

IBM ID Doc considers the behavior of hyperlinks to be part of the information presentation style and has specific language constructs in the IBM ID Doc style language for defining link style. Aspects of link style include how link anchors should be represented and what action to take when links are activated or traversed.

IBM ID Doc hyperlink function is fully HyTime compliant. IBM ID Doc uses the location address module, requiring support for the optional notloc location address element (see “Notloc (notation-specific location)” on page 232).

IBM ID Doc processors can use both explicit and implicit hyperlinks to derive various presentation styles without the need for explicit positioning by the author. For example, in a print style where explanatory notes are placed in the offset margin next to the information they annotate, all the author has to do is associate the note with the annotated information. This can be done one of three ways.

The simplest way to link a note to, for example, a paragraph is to make the note part of the paragraph's content, like so:

© Copyright IBM Corp. 1992, 1993

```
<prolog>
  ⋮
<PropertyDefinitions>
<Classdef ElementName=note
          ClassName=ExplanatoryNote
          SemanticClass=Annotation>
<desc>Explanatory notes are notes that explain or amplify the paragraph
or other element they are linked to.  The default print presentation is
to place the note next to the paragraph in the outer margin.
</Classdef>
  ⋮
</PropertyDefinitions>
</prolog>
  ⋮
<p>
<note class=explanatorynote>
This is an explanatory note.
</note>
This is the paragraph text.
```

By being contained by the paragraph, the note is implicitly linked to the paragraph. Using the link style language, you can define a presentation style for notes of class ExplanatoryNote such that they are placed next to the element that directly contains them.

The hyperlink relationship between the note and the paragraph is all the information an output processor needs to properly position the note next to the paragraph. There is no need for the author to define the positioning in any way. The positioning of the note represents the hyperlink relationship between the paragraph and the note. In addition, by defining a note class of ExplanatoryNote and associating a link and presentation style with it, you have defined a precise and meaningful semantic to the effect that explanatory notes are annotations of their containers (in this case a paragraph) and should be presented in a way that represents that relationship, such as in the margin next to the paragraph. In an online system the annotation relationship might be represented by a special icon that could be selected to bring up the note or by the action of having the note appear in a separate window when the paragraph is viewed.

Because the logical relationship between the note and the paragraph has been defined by the author, that relationship can be expressed in a variety of ways.

IBM ID Doc defines three basic classes of information, called “semantic classes” because each class defines the base meaning, or semantic, associated with the information. The three classes are content, metadata, and annotations.

Content is anything that isn't metadata or annotation. Metadata is data that describes the element that contains it. For example, titles are always metadata because they describe the element that contains them.

Annotations are elements that describe the content of another element. For example, footnotes are always of semantic class annotation, because they describe or annotate the content. In IBM ID Doc, all elements of semantic class annotation are implicitly linked to the element that contains them. Processing systems can use this linkage to determine how to represent and present annotation elements. Annotation elements can also explicitly link to other elements in order to annotate them using the Linkend= attribute.

Elements can also link to annotations explicitly using the common Annotations= attribute. Any type of element can be linked to in this way. The element linked to becomes an annotation of the linker, regardless of the linked element's semantic class.

Finally, using the common ClassDef element, you can define classes for any element and define the semantic class of the new class. For example, in the explanatory note example, the class ExplanatoryNote would be defined like so:

```
<Prolog>
 <ClassDef ElementType=note
           ClassName=ExplanatoryNote
           SemanticClass=annotation>
  <desc>Notes of class ExplanatoryNote are to be used for explanatory
notes associated with
paragraphs or paragraph blocks.  The default
presentation should be to present the note in
the offset column next to the paragraph.
 </ClassDef>
</Prolog>
```

The specifics of the presentation of explanatory notes would be defined using the IBM ID Doc style language or the style definition mechanism for the processor you are using (for example, an online presentation system may have its own style definition function).

Direct or Explicit Links

IBM ID Doc defines a number of different explicit links types for various purposes, including the use of another element's content. All explicit links use ID= attribute values to connect two elements together. IBM ID Doc provides a method for indirection via the LinkDesc element. The indirection provided by LinkDesc lets you do things like define associative or query links or assign a single ID to a list of link targets. The LinkDesc element associates standard HyTime location elements with a Desc element, which is used to describe the purpose of the link. HyTime location elements are also valid in some contexts outside of LinkDesc.

Each of the different explicit links is explained in the following topics.

Cross Reference Links

Cross reference links are defined using the generic XRef element. The processing semantic for cross reference links is that a reference to the link target is generated. The generated reference can contain the target's title or short title, its number (if is an enumerated element), or its locator (typically a page number in print or a topic number on line). The specifics of the cross reference style are defined by the active presentation style.

Content Reference Links

There are times where it is useful to be able to re-use an element in more than one place. For example, in describing a user interface, the same menu selection field might occur in the descriptions of many panels. IBM ID Doc lets you re-use elements by providing a content reference link. For any element, you can specify a ContentRef= attribute that points to another element of the same type. The content of the linkee will be used at the point where the linker is specified. In the following

example, the second DLEntry uses a ContentRef= link to get the content of the first DLEntry in the second list:

```
<InfoModule>Panel 1
  ⋮
 <parmlist>
  <parmlEntry id=FilePulldown>File
   <pd>Use the File pulldown to...
  </Parmlist>
  ⋮
<InfoModule>Panel 2
  ⋮
 <ParmList>
  <ParmlEntry ContentRef=Filepulldown>
  <ParmlEntry>
  ⋮
```

The result of the ContentRef= link will be the same as if the first ParmlEntry had been copied into the second list.

Note: Support for content reference links is optional.

Retrieval Alternatives

Retrieval alternative links identify elements that are alternatives to the linker that can be used in addition to or as a replacement for the linking element. For example, you can create translated versions of a paragraph and then link the translations to the main paragraph using the RetrievalAlts= attribute:

```
<nameloc id=para1-alts>
 <nmlist>para1-FR para1-Ger</nmlist>
</nameloc>
<p id=para1 language="USEng"
   Linkends  ="para1-alts" HyTime=ilink
   AnchorRoles="linksource alternate">
```

Whether or not a retrieval alternative for an element is used is a function of the active retrieval conditions, which are set at the time document processing is started. For example, you may set the active language so that only elements in that language are retrieved.

Property Reference Links

Elements can refer to other elements in order to inherit their property values using the PropertySource anchor role to identify property reference links. Property reference links are normally used to link to xDEF elements contained with a PropertyDefinitions element, but they can be used to link to other elements as well. Properties inherited by direct reference take precedence over properties an element inherits from its parents. Properties specified directly on an element take precedence over all other elements. See “Property Inheritance” on page 18 for more on property inheritance.

Generic Hyperlink

All elements can take the LinkEnds= attribute. LinkEnds= represents a hyperlink for which IBM ID Doc does not define a specific meaning or semantic. You must use the link style definition language to define the meaning of LinkEnds= links.

Linking to Other Documents

When cross-document links are supported, you can use any of the explicit hyperlink mechanisms to link to elements in other SGML documents by using the Docname= attribute on a NMList element within a Nameloc element. The Docname= attribute contains the name of a SUBDOC or other entity that defines the document that contains the target element.

The entity name is then used on the DOCNAME= attribute like so:

```
<p>See
<nameloc id=xref-target>
 <nmlist docname=IBMiddocref>anidvalue</nmlist>
</nameloc>
<xref refid=xref-target> for more information.
```

In this example, a reference will be created to the division with an ID= value of 'anidvalue' in the *IBM ID Doc Reference*. The XRef element refers to the Nameloc element, which contains a NMList element. The NMList element then associates the Docname= value with the ID of the element within the target document. This is the standard HyTime mechanism for creating cross-document hyperlinks.

Using HyTime Location Elements with Explicit Links

IBM ID Doc lets you create arbitrary hyperlinks directly from point to point. For example, the following Link element creates a hyperlink to the division following it:

```
<p>Use the <link linkends="fubar-func">fubar function</link> to do ...
  ⋮
<div id=fubar-func>The Fubar Function
  ⋮
```

This is a relatively simple link, and is the appropriate way to create hyperlinks when a given target is only referred to a few times. However, if the target is the subject of many links, it is a good idea to use an indirect method of linking to protect the linkers from changes to the target (for example, the target's ID changing).

To create indirect links, you use HyTime location elements to associate a local ID with the ID of the actual target. These location elements are contained within a LinkDesc element, which associates a description with them:

```
<LinkDesc>
 <desc>Location of Fubar Function explanatory information.
 <nameloc id=fubar-func-expl>
  <nmlist>fubar-func
 </nameloc>
</LinkDesc>
```

The Nameloc element is the cornerstone of the HyTime location elements. Nameloc then contains either a NMList (name list) element, which then contains a list of either element IDs or entity names, or a NMQuery (name query) element,

which contains a query that returns the names (IDs or addresses) of elements that satisfy the query.

In this simple example, the Nameloc element is used to provide one level of indirection between references to information about the fubar function and the specification of the actual location of that information.

While this may seem like a lot of extra work, consider the case where there are tens or hundreds of links to the explanation of the fubar function. If something about that information changes, say it is moved to another document or the ID changes or it is split into several pieces, then if each of those 100 links pointed directly to the information, you must then find and change all 100 links. However, if you have used the indirect method shown above, you need only change the value contained in the Nameloc element—the linking elements themselves need never change.

The rule of thumb developed after several years experience with BookMaster hyperlinks is that creating the indirection is worth effort if a given target is linked to more than twice. Note that it is relatively easy to generate the LinkDesc and Nameloc markup using editor macros. Also, over time, SGML editors will provide the function of managing this sort of indirection automatically so that you as an author don't have to worry about it.

Another advantage of indirect linking is that it allows you to use queries to create hyperlinks or link to multiple targets from a single link point. For example, instead of linking directly to the section on the Fubar function, you could use a query to find all information about Fubar available at the time the link is traversed:

```
<LinkDesc>
 <desc>Location of Fubar Function explanatory information,
 found by query.
 <nameloc id=fubar-func-expl>
  <nmquery notation=myquery>Find "fubar function" within (<title> or <lename>)
 </nameloc>
</LinkDesc>
```

Here, instead of a NMList element, the Nameloc contains a NMQuery element which then contains a query, in this case in a made up notation. A link to the Nameloc element is really a link to all the elements that satisfy the query. Using queries for hyperlinking can make your hyperlinks both powerful and easy to maintain because as long as the information exists and is available, the link will work, regardless of precisely where the information is or what form the information it is in.

You can also use Nameloc and NMList to create hyperlinks to entities rather than elements. In the usual case, you create hyperlinks to NDATA entities such as graphics or video clips. For example, you might do the following:

```
<LinkDesc>
 <desc>Define a link to a video by reference to an entity.
 <nameloc id=fubar-tutorial>
  <nmlist entity>fubarvideo
 </nameloc>
</LinkDesc>
```

In this case, the NMList element is defined as containing entity names. The entity shown in the example (fubarvideo) might be defined like so:

```
<!DOCTYPE IBMIDDOc PUBLIC "+//ISBN 933186::IBM//DTD IBMIDDoc//EN" [
 <!ENTITY fubarvideo SYSTEM "fubartut.avc" NDATA AVCVideo
    -- Tutorial video for the Fubar function -->
]>
```

IBM ID Doc only requires the base location module of HyTime, and only supports Nameloc and NMQuery. However, the full set of HyTime location elements provides a very rich set of mechanisms for locating just about any sort of data or object your computer system is capable of accessing. The use of more advanced HyTime location functions is simply an extension of the indirection provided by Nameloc. As more HyTime function is added to your presentation or processing system, you should be able to create more sophisticated hyperlinks by building on the basic indirection methods shown here.

Finally, note that in IBM ID Doc, any hyperlink can make use of HyTime location elements to define locations, not just the Link element. Thus, for example, you could use Nameloc with XRef to provide more protection for important cross references.

Qualified Element IDs

IBM ID Doc is designed to enable the re-use of document fragments. The re-use of document fragments poses a basic problem of ID management in the case where a single document fragment is used in several different documents. If that fragment contains IDs, someone or something must ensure that the IDs in the re-used fragment are not the same as any IDs in the documents in which it is included. This requirement comes from the fact that in SGML, each SGML document is a single ID (and entity) name space within which all IDs must be unique.

It is clear that trying to manage IDs in re-used external text entities is practically impossible because it means that there is essentially a single name space across all documents, which is impractical and unnecessary. Fortunately, SGML and HyTime provide a solution.

IBMIDDoc solves the problem of re-using document fragments by making use of the SUBDOC feature of SGML and the cross-document hyperlinking function defined by the HyTime standard. IBMIDDoc also defines a few simple constraints on the use of subdocuments that ensure that their use will not break the processing systems.

In IBMIDDoc, we make the distinction between the “logical” document and a “syntactic” document. A syntactic document is a complete SGML document according to the rules of ISO 8879, e.g., a doctype declaration plus document instance. A logical document is the set of elements (or data objects) that the *processing application* considers to be a complete document, which in the case of IBMIDDoc is the set of elements consisting of a prolog, a body, and optional front and backmatter elements. This distinction is possible to make because the application never sees the actual syntactic encoding of an SGML document—it only ever sees the data contained in the SGML as passed to it by the SGML parser. Given this, it is possible to use subdocuments to organize a document's data into multiple syntactic documents that make up a single logical document.

The SGML subdocument feature provides a mechanism for refering to complete syntactic documents from within another syntactic document with the constraint that

the subdocuments must conform to the same SGML declaration as the main document. ISO 8879 does not define any other restrictions, which allows some clearly non-sensical cases, such as refering to a subdocument from within a paragraph or something. Therefore, the IBMIDDoc design defines additional constraints on the use of subdocuments that make their use practical and reliable.

The IBMIDDoc application defines the following rules for the use of subdocuments for the reuse of document fragments[6]

1. A subdocument entity must be proper subset of the main document type. In other words, it must use the same public DTD as the main document and it must consist of a subelement of the IBMIDDoc element (e.g., Div, ObjectLibrary, P, etc.). This can be easily accomplished without modifying the public part of the IBMIDDoc DTD simply by naming the document type appropriately:

   ```
   <!DOCTYPE Div PUBLIC "+//ISBN 0-933186::IBM//DTD IBMIDDoc//EN">
   <div>Title of the Div
   ```

   The rules of SGML ensure that any elements declared in the DTD but not used in the document element's content model will be ignored.

   The subdocument can be any valid IBMIDDoc element, right down to Phrase, allowing any granularity of re-use desired.

2. Subdocument entities can only be refered to via the common ContentRef= attribute; direct inline references to subdocument entities will be ignored. When the unified name space option of HyTime is used (which is the default and is highly recommended), the entity name can be specified directly as the value of the ContentRef= attribute. When the unified name space option is not used, you must use the normal Nameloc/NMList mechanism to convert an ID reference into the necessary entity reference, for example:

   ```
   <div contentref=common-div-1>
   <nameloc id=common-div-1>
    <nmlist entity>Common-Division-One</nmlist>
   </nameloc>
   ```

3. The document type of the subdocument must be the same as the generic identifier (tag name) of the element from which it is referenced. This restriction ensures that the data in the subdocument is what the processing application expects at that point. This very conservative restriction can be relaxed in either of two ways, if desired:

   a. The subdocument document type must be of the same architectural form(s) as the element from which it is referenced.

   b. The element contained by the subdocument document type is the same as the element doing the referencing.

This use of subdocuments makes subdocuments the unit of blind re-use. Normal external text entities, if used, cannot be reliably interchanged.

---

[6] There are other uses of subdocuments, such as the inclusion of data using completely different DTDs, that are not re-use in sense defined by IBMIDDoc. In these cases, the SGML documents should be declared as data entities with a specific notation, rather than as subdocuments. Defining these entities as data entities with notations makes clear to humans and to processing systems that this data cannot be processed using the basic IBMIDDoc processing.

This use of subdocuments also enables the local definition of new elements in re-used subdocuments by binding the new element declarations to the context in which they are used (see Chapter 3, “Extending The IBMIDDoc Document Type Locally” on page 22).

The main complication of this use of subdocuments is that most cross references or hyperlinks will become cross-document hyperlinks, where the documents being crossed are syntactic SGML documents, not logical documents. However, the same standard HyTime mechanism used to enable hyperlinking between two logical documents is used to do hyperlinking between and among subdocuments. It is recognized that it is not practical for authors to manage this indirect markup without dedicated authoring tools and we assume that authors will have the necessary authoring support before being asked to use this mechanism (although it's really not that hard to code up in practice). We expect that authors will use data management systems that will manage the re-use of logical document fragments and hyperlinks to them automatically.

Property Inheritance

IBM ID Doc defines a number of properties that can be associated with elements, such as language, status, classification, and so on. IBM ID Doc also defines an inheritance semantic by which elements can inherit property values from other elements.

All elements inherit their parent's properties. For example, a paragraph within a division that has a language property value of Spanish will inherit that property and will thus be identified as Spanish as well.

Elements can also inherit properties from other elements by referring to them with the common PropertyRef= attribute. Properties of elements referred to with PropertyRef= always take precedence over properties inherited from parents. Properties specified directly on an element always take precedence over inherited properties.

In addition to property inheritance, some properties may be implied by value of an element's children. For example, an element's security classification is the highest classification of any of its children.

Using The PropertyDef Element

The PropertyDef element provides a general mechanism for defining a set of property values once and then using those properties by reference from other elements. For example, you may have a complex set of Version= and Level= specifications that will apply to a number of elements. Rather than copying the complex specification onto every element that needs it, you can specify it once on a ProperyDef and then use the PropertyRef= attribute to refer to the PropertyDef element, as shown in the following example.

```
<prolog>
<propertyDefinitions>
<PropertyDef id=v1not2
             version='(version1 #AND #NOT version2)'>
</PropertyDefinitions>
</Prolog>
<body>
<div linkends   ="v1not2" HyTime=ilink
     anchorroles="linksource propertysource">
  <title>For Version 1 and not Version 2</title>
<p>
  ⋮
```

In this example, the PropertyDef element is given an ID value, 'v1not2', and a version specification of some complexity. On the Div element, the version property is inherited by using the property reference role to refer to the PropertyDef element by specifying its ID value. Thus, the division will only be used when the active version is 'version1'.[7]

Using The ClassDef Element

The ClassDef element lets you define classes for IBM ID Doc elements in order to make the markup in your document more precise in order to enable processing functions such as generation of detailed glossaries or bibliographies, precise associative links, or presentation effects unique to your information. Author-defined classes can also be used to control inheritance of element properties.

In most production environments, element classes are defined for an entire collection of documents by some person responsible for the design of the information in the collection, such as a library planner or documentation specialist. Under normal circumstances, individual authors should not have to worry about defining element classes and defining class-based property inheritance. If a library planner is defining classes for you, all you need to know are the class names and the elements to use them on. You do not need to understand how classes are defined.

The typical use of classes is to define specific phrase classes that reflect the specific information you are writing about. For example, say you are documenting software that uses two unique constructs, Whatsits and Thingies, constructs unique to your product and not accounted for in the base IBM ID Doc language. You want to be able to do things with references to Whatsits and Thingies, such as apply specific presentation styles to them, define associative link behaviors, and enable searching for occurences of references to Whatsits and Thingies. You'd also like to have all Whatsits and Thingies indexed for you automatically.

You can enable all these functions by defining an element classes and Whatsits and Thingies and then using Phrase elements with those classes to identify references to them.

The first step in defining new element classes is to decide what the class names should be. This is an information analysis task. You analyze what it is you are

[7] What property values are active when you process a document is a function of the processing system you are using. In typical systems, you will either define the active property values using an input panel or you may use some sort of options file to define the properties. See the documentation for your processing system to see exactly how active properties are defined.

writing about to determine what kinds of things there are. In this example, we have analyzed our product and determined that there are two important types of objects, Whatsit objects and Thingy objects.

The next step is to define exactly what these things are so that we are clear when and why to use them.

You use the results of your analysis to define element classes using ClassDef elements, as shown in the following example. ClassDef elements are only valid within PropertyDefinitions, which goes in the document prolog or a division prolog. Since in this example the classes will apply to the entire document, they should be put in the document prolog:

```
<IBMIDDoc>
<Prolog>
  ⋮
<PropertyDefinitions>
<ClassDef classname=whatsit>Whatsit Objects
 <Semantic>Identifies whatsit objects.  Whatsits are hardware
 components.
</ClassDef>
<ClassDef classname=thingy>Thingy Objects
 <Semantic>Identifies thingy objects.  Thingies are software
 components.
</ClassDef>
</PropertyDefinitions>
  ⋮
</prolog>
```

Within the ClassDef element, the Semantic element contains a description of what the class means. The purpose of the Semantic element is to force you to think about the classes you are defining and to remind you that classes should be meaningful. Classes should describe real things or aspects of real things. Classes should not relate purely to processing or presentation effects. For example, as class of 'bold' is probably not meaningful. As a rule, class names should be nouns.

Having defined the above classes, we can use the class names with the Phrase element to identify names of Whatsits and Thingies in our document:

```
<div>Hardware Problems
<p>Hardware problems are usually caused when
the <phrase class=whatsit>famtoozler</phrase>
gets out of adjustment.
Readjust it using the <phrase class=thingy>famtoozlometer</>
component of the &prodname&semi; analyzer.
```

Having defined classes and used them on elements, you can use the classes to define property inheritance for elements of a given class. For example, say you want the names of whatsits to be presented in bold and thingies to be presented in small capitals. You can do this by using PropertyDef elements to define the presentation style parameters and then refer to the PropertyDef from the ClassDef element. Using the example above, you would do this:

```
<IBMIDDoc>
<Prolog>
  ⋮
<PropertyDefinitions>
<PropertyDef id=boldstyle
             style="bold">
<PropertyDef id=smallcapsstyle
             style="smallcaps">
<ClassDef classname=whatsit PropertyRef=boldstyle>Whatsit Objects
 <Semantic>Identifies whatsit objects.  Whatsits are hardware
 components.
</ClassDef>
<ClassDef classname=thingy PropertyRef=samllcapstyle>
Thingy Objects
 <Semantic>Identifies thingy objects.  Thingies are software
 components.
</ClassDef>
</PropertyDefinitions>
  ⋮
</prolog>
```

Any Phrase of class Whatsit will now inherit the property style defined on the PropertyDef element refered to by the ClassDef element. You could also define presentation styles for classes using the IBM ID Doc style language in a separate style document.

The use of class names on the phrase elements improves the retrievability of your information by enabling precise searching and associative linking, as well as automatic indexing.

Precise searching is enabled for online presentation systems that preserve the structure of the original document markup. In these sorts of systems, a reader can specify a search such that only elements of a given class will satisfy the search. For example, while viewing your information, a reader could find all references to whatits by specifying the search “find all elements with the class= value 'whatsit'”. This will have the effect of finding all whatsits identified as such in your document.

Associative linking is enabled because an online presentation system can, for example, associate phrases of a certain class with other elements that have the same class **and** contain the text of the phrase. For example, say in your document you define all the different whatsits on a definition list. If you assign the class whatsit to that definition list, an online presentation system can automatically associate any phrase of class whatsit to that definition list, and only to that definition list. The definition list would look like this:

```
<dl class=whatsit><title>Definition of Whatsits
<dt>famtoozler
<dd>A thing that famtoozles
<dt>whantingler
<dd>A thing that whantingles
</dd>
```

Chapter 3. Extending The IBMIDDoc Document Type Locally

The IBMIDDoc design is intended to allow authors to extend the identification power of specific elements to more precisely match the things they're writing about. IBMIDDoc provides two mechanisms for doing this. The first mechanism is the common CLASS= attribute, by which authors can define classes specific to their information and use those classes on general elements. This mechanism, while easy to implement, is not the best mechanism to use since SGML already defines a mechanism for defining element classes, namely generic identifiers defined on element declarations.

However, most existing SGML systems have no mechanism for managing the ad-hoc definition of new elements, as most processing is based off of generic identifiers. IBMIDDoc, however, being architecturally based, has a solution for this problem that offers the potential for authors to extend the DTD locally to a document or subdocument in a reliable and managable way.

IBMIDDoc's solution is to impose constraints on what authors can declare and to continue to use the class definition mechanism as a structured way for authors to both document what they've done and place their new elements in a defined semantic context.

The restriction imposed is very simple: locally-defined elements must conform to one or more known architectures, where the architectures are known by the processing system, e.g., InfoMaster, HyTime, IBMIDArch, etc.. The newly-declared elements must also have a fixed CLASS= value that refers to a ClassDef element either in the local document or in a central class definition document. Within these restrictions, authors are free to do almost anything they need to do. Given the generality and flexibility of the InfoMaster and IBMIDArch architectures, authors should be able to satisfy the needs of almost any information using this mechanism.

Note that it may still be necessary for authors to provide special case processing or style definitions for their new elements, depending on the requirements of the information and the output processes being used. There is no way to avoid this requirement, but the use of architecture-based defaults for style and processing should ensure usable, if not optimal, results in all cases.

Local definition of elements can be coupled with the use of subdocuments for creating re-usable logical document fragments (see “Qualified Element IDs” on page 16).

© Copyright IBM Corp. 1992, 1993

Part 2. Language Reference

Chapter 4. Introduction 24

Chapter 5. The IBMIDArch Architecture 30

Chapter 6. IBM ID Doc Modules 32

Chapter 7. IBM ID Doc Elements 35

© Copyright IBM Corp. 1992, 1993

Chapter 4. Introduction

This document is the definitive description of the the IBMIDDOC application of the InfoMaster Architecture.

Overview of the Reference Entries

The reference entries have the following structure:

ElementName
Short Description

Purpose
Describes the purpose of the element; its “semantic”.

Syntax
Contains a syntax definition that describes the rules for typing the element markup in a text editing environment, such as Xedit or EPM. See “How to Read the Syntax Diagrams” for details on reading the syntax diagrams.

Attributes and Contained Elements
Contains an explanation of any attributes or contained elements.

Usage
Describes how to use the element.

Examples
Contains one or more examples of how the element might be used.

SGML Definition
Contains the formal SGML definition of the element. The information in this section is exactly the same data that is used to create the actual IBM ID Doc DTD. In fact, the DTD fragments presented in this document are the same source used to generate the production DTD.

Valid Contexts
Lists any elements related to this element. An element's valid parents are always listed first.

How to Read the Syntax Diagrams

The syntax diagrams for the elements define the rules for typing element markup in text editing environments such as Xedit and EPM. The syntax diagrams will not show contained elements that have start and end tags that are completely omissible and are thus not normally specified explicitly. (Note that when using SGML editors, such as IBM TextWrite or Author/Editor, you will generally see representations of all elements, including omissible elements.)

The following conventions are used in the diagrams:

© Copyright IBM Corp. 1992, 1993

- All keywords, including tag and attribute names, are shown in upper case. They may be entered in any case.
- Single-word attribute values are not shown with quotes (but any attribute value can be entered with quotes around it). Quotes are always shown as double quotes ("), but single quotes (') are also valid.
- For simplicity, completely omissible tags are never shown.
- Fragment references are used to indicate references to contained elements defined separately.
- The following fragment names have special meaning:

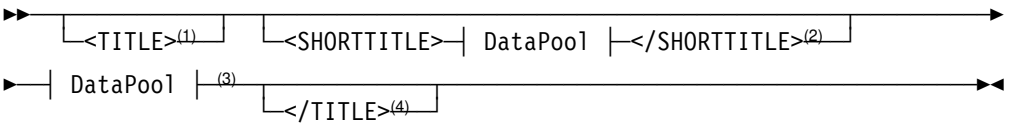
  Common Attributes
  Refers to the common attributes valid on all elements. The common attributes are defined in "Common Attributes" on page 77.

  DataPool
  Refers collectively to the phrase-level elements and normal character data. See "PhraseLevelElements" on page 256.

  InfoUnitPool
  Refers collectively to the paragraph-level elements. See "ParaLevelElements (paragraph-level elements)" on page 245.

  InfoPool
  Refers collectively to DataPool and InfoUnitPool together. See -- Heading 'INFOPL' unknown --.

The following diagram is a typical example. The syntax notes highlight the various aspects of the diagram.



Notes:

1. The element start tag, indicating, in this case, that the start tag is optional in some circumstances.
2. Here, the ShortTitle element is indicated as being optional, but when specified, the start and end tags must be specified.
3. A reference to the fragment "DataPool", indicating that normal characters and phrase elements are valid within Title.
4. The element end tag, in this case, optional. End tags are always shown unless it is an error to specify an end tag, as for the XRef element.

How to Read the SGML Definitions

The reference entry for each element and attribute contains the formal SGML definition of the rules for that element or attribute. The SGML definition defines the complete rules for specifying elements and their content, in contrast to the syntax diagrams, which define the minimal rules for typing in tags and attributes. You may need to refer to the formal SGML definition if you are using an SGML editor that always represents all elements, even those that are completely omissible (and thus not shown in the syntax diagrams).

If you are creating an IBM ID Doc processor, you must use the SGML definitions to understand exactly what the rules are for each element. The SGML definition is the definitive definition of the rules for an element or attribute.

SGML elements are defined with two SGML declarations: element declarations and attribute list declarations. Element declarations define the allowed content for an element and the start and end tag minimization rules. Attribute list declarations define the attributes valid for a given element and the content rules for the attributes.

Element Declarations

A typical element declaration looks like this:

```
<!ELEMENT Title   O O (ShortTitle?, TitleBody)  -- Title element -->
```

The first word following '<!ELEMENT' is the element name, in this case Title. The next two characters indicate the start and end tag minimization, either a dash (-) for required, or the letter O for omissible. In this case, both the Title start and end tags are ommissible.

The next part defines the element content model, either one of the keywords EMPTY, CDATA, or RCDATA, or a group of element names or the keyword #PCDATA within parenthesis. EMPTY means that the element contains nothing. CDATA means that the element contains only character data (no tags or entity references), and RCDATA means that the element contains only replaceable character data (entity references but no tags).

Element names within parenthesis are called a *name group*. Within a name group, #PCDATA indicates zero or more normal characters. Everything else within a name group must be the name of another element, possibly modified by an occurrence indicator, and connected with one of the allowed connectors. Within a name group, element names can be further grouped within parentheses to form a logical unit called a *syntactic token*. The occurrence indicators define whether or not an element or group is required and whether or not it can be repeated. The connectors indicate whether or not elements must be specified in a certain order.

The occurrence indicators are:

' ' No occurrence indicator indicates that an element is required and can only be specified once.

'?' A question mark indicates that an element is optional and can only be specified once if specified at all.

'*' An asterisk indicates that an element can be specified zero or more times (in other words, it is optional and repeatable).

'+' A plus sign indicates that an element can be specified one or more times (in other words, it is required and repeatable).

Thus, in the example above, the ShortTitle element is optional and the TitleBody element is required.

The connectors are:

',' Comma is the *sequence indicator* and indicates that elements must be specified in the order they occur in the content model name group.

'|' The OR bar indicates that any one element in a group can be chosen on any pass through the group. When an OR group is repeatable, the same element can be specified multiple times and elements can be specified in any order. Almost all groups in IBM ID Doc are repeatable OR groups.

'&' Indicates that all elements in a group must be specified, but that they can be specified in any order. IBM ID Doc does not use any AND groups.

Attribute List Declarations

A typical attribute list looks like this:

```
<!ATTLIST Link
    RefIDS           NAMES     #REQUIRED
    LinkPhrase       (auto)    #CONREF
    %common.atts;

    InfoMaster       NAME      #FIXED  "HyperLink"
            -- HyTime attributes: --
    HyTime           NAME      #FIXED clink
    reftype          CDATA     #FIXED "#ALL #ANY"
            -- No REFLEVEL restrictions (see 10744 6.5.3) --
    refrange         CDATA     #FIXED "#ALL I" -- indirection --
    HyNames          CDATA     #FIXED "RefIDS LINKENDS "
>
```

The first word following '<!ATTLIST' is the name of the element the attributes are for, in this case the Link element. Each line following defines a single attribute (some attribute definitions may be split across two or more lines; line breaks within declarations are not significant to SGML).

For each attribute, the first word is the attribute name. If the next thing is a word, it is an SGML-defined keyword defining the data type of the attribute, for example NAME, which indicates that the attribute must contain only characters valid within SGML names. Following the datatype keyword is one of the following keywords:

#REQUIRED Indicates that the attribute is required and must be specified by the author.

#IMPLIED Indicates that the attribute is optional, and if not specified the attribute value will be provided by the processing application.

#CONREF Indicates that when specified, the element's content is generated by the processing system. When a #CONREF attribute is specified, it is an SGML error to specify the element end tag. In IBM ID Doc, almost all elements can take the ContentRef= attribute, which is a #CONREF attribute.

#FIXED Indicates that the attribute value is fixed and cannot be changed by an author. If a #FIXED attribute is specified, its value must exactly match the value defined in the DTD.

If the part following the attribute name is a group of names enclosed in parenthesis, it indicates the valid values for an attribute. Attributes of this sort are called *value attributes*, and they may be specified by simply specifying one of the values, rather than `attname=value`. In the case of the Link element, the LinkPhrase attribute is a value attribute with one valid value, 'auto'. Thus, an author can specify the Link

tag like so: `<link refids=target1 auto>`. Value names within a group can be separated by commas or by OR bars (|).

If there is a word following the value group, it indicates the default value. If there is no word following the value group then the attribute's value is implied if not specified.

The keyword #FIXED is always followed by the attribute value specification, which defines the attribute's fixed value. In IBM ID Doc, fixed attributes are used to define conformance with various architectures, including the HyTime standard and the InfoMaster architecture. Fixed attributes are also used to define the content rules for other attributes and for elements that have #PCDATA content. These attributes and their use are all defined by the HyTime standard.

In the IBM ID Doc definitions, the first set of attributes are those attributes that an author can specify, with the last one always being a reference to the common attributes parameter entity, which contains the definition of all the common attributes. Following the reference to %common.atts; are any attributes that have fixed values.

How to Read HyLex Specifications

The HyTime standard defines a notation called *HyLex* for defining the rules for specifying character strings. IBM ID Doc uses HyLex notation to define the content rules for some attributes and for the content of some elements that have #PCDATA content.

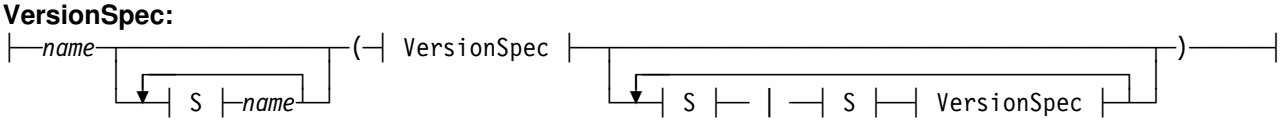
HyLex is an extension of the normal SGML content model notation that uses literal strings and references to pre-defined HyLex models rather than element names. The same occurrence indicators and connectors are used in HyLex as in SGML content models.

Within a HyLex expression, unquoted names refer to HyLex lexical types defined elsewhere, usually in the HyTime standard. When IBM ID Doc defines new lexical types, the types are defined at the place where they are used. A typical HyLex expression looks like this:

```
((name, (S, name)*) | ( "(" VersionSpec, (S, "|", S, VersionSpec)*, ")"))
```

For comparison, the equivalent syntax diagram for this expression is:

VersionSpec:



Within the context of an attribute list definition, HyLex expressions occur in LexModel #FIXED attributes. The value of the LexModel attribute is one or more attribute name/HyLex expression pairs, where an attribute name is either the name of an attribute in the element's attribute list or the keyword #CONTENT, indicating the element's #PCDATA content. A HyLex expression is either a single lexical type name or a more complex HyLex expression contained within parenthesis. The LexModel definition of the LexModel attribute is thus:

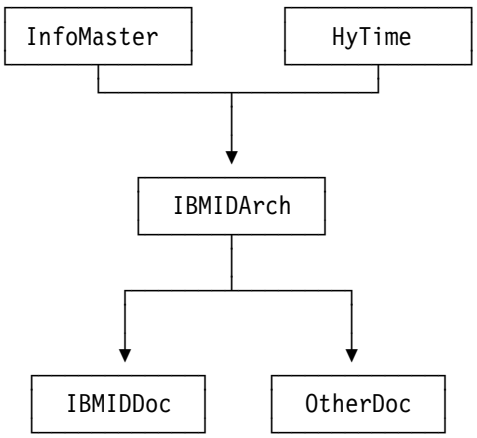
```
<!ATTLIST element
    LexModel  CDATA #FIXED
      "LexModel ((AttributeName, S, (ltn | ('(', HyLexExpr, ')'))*)"

    -- ltn is "lexical type name" as defined by HyTime --
    -- HyLexExpr is any valid HyLex expression  --
>
```

Chapter 5. The IBMIDArch Architecture

The IBMIDArch Architecture is an application of the InfoMaster Architecture that defines the base elements of the IBMIDDoc application as architectural forms. From the IBMIDArch architectural forms can be derived concrete elements with the same properties and constraints as their forms, but with unique semantics or additional properties.

These layers of architecture can be thought of as an abstract object model as in object-oriented programming, with the object hierarchy being:



The chief use of the IBMIDArch architecture is to enable the easy definition and implementation of new specialized modules on top of the IBMIDDoc base, but it can also enable the creation of or interoperation with other document types that have different specific applications from IBMIDDoc but use the same base architecture.

As for InfoMaster and HyTime, IBMIDArch forms are declared using a special attribute, the IBMIDDArch= attribute, to associate an architectural form name with a concrete element in a given DTD.

Note also that as for HyTime, only document instances must conform to the architectural form constraints. The DTD in which architectural form conformance is declared may allow instances that do not conform to the architectural form declarations.

It is expected that the IBMIDArch architecture will be primarily used as guide to system implementors who wish to design layered systems, possibly using object-oriented methods. It is strongly suggested that the concrete objects used in implementing IBMIDDoc and IBMIDArch processors reflect these architectures literally. It is presumed that in such a system implementing support for new elements derived from architectural forms would be done by deriving subclass objects from the program objects that represent the architectural forms. Note also that it is not necessary to actually implement direct support for the IBMIDArch architecture in an conforming IBMIDDoc application. In other words, there is no requirement to provide any processing specifically for IBMIDArch architectural forms, although it is

© Copyright IBM Corp. 1992, 1993

likely that any non-trivial implementation will do so in any case because that is the natural way to do the implementation.

The IBMIDArch architecture is not defined separately from the definition of the base elements in the IBMIDDoc language. Elements whose IBMIDArch= attribute value is the same as their GI are, by definition, also IBMIDArch architectural forms, and their content constraints and properties (attributes) define the constraints of the architectural form as well.

Chapter 6. IBM ID Doc Modules

The IBM ID Doc application is organized into a base set of elements and required processing functions plus a number of optional specialized modules defining elements for specific purposes. In addition to the optional modules, IBM ID Doc includes a number of optional processing features that implementations can choose to support if desired. The intent of this organization is to make is possible to create low-cost implementations of IBM ID Doc processors, while providing a mechanism for extending the application as needed.

Module and Function Use Declaration

IBM ID Doc follows the HyTime model and thus uses the APPINFO statement in the SGML declaration to define the base support level, while using processing instructions to declare support for optional features.

Note to Wayne

We need to work out the details here, I assume on the model of InfoMaster

Organization of IBM ID Doc Modules

The IBM ID Doc application consists of the following modules:

Base The base module defines those elements common to all modules.

LERS Language Element Reference Sections. This module is a special case of Modular Information. It is intended for creating computer language reference information.

Messages and Codes Modular information for documenting messages and codes issued by computer systems.

Syntax definitions Syntax definitions are used to define in detail the syntax of computer language statements or similar structures. The intent of the syntax definitions elements is both to enable the automatic creation of graphic representations of the syntax and to enable the use of the definitions by automated systems such as command builders or command parsers.

Procedures Procedures are a specialized form of modular information intended to enable the complete and precise documentation of step-by-step procedures, possibly in sufficient detail that the procedure descriptions could be used as the knowledge base for expert systems or diagnosis aids. Procedures include elements for defining conditional branching among procedures.

Hypertext webs The Web elements are intended to enable the creation of documents that have no inherent linear structure, but rather are completely defined in terms of explicit hyperlinks among independent web nodes and possibly associated content objects. Webs are a way of explicitly representing unstructured hypertext.

© Copyright IBM Corp. 1992, 1993

Full HyTime support The full HyTime support module provides instances of all the elements defined in the HyTime standard, ISO 10744, including all location element types as well as the elements needed for defining finite coordinate spaces and event schedules.

Man pages Man pages are special case of modular implementation that implement the Davenport Committee for the Common Man Man page structures.

XHelp The XHelp module provides support for the XHelp DTD defined as part of the XHelp protocol proposal.

IBM-specific elements Some elements have IBM-specific versions in order to meet the specific needs of IBM processes or products. All IBM-specific elements start with 'IBM', for example, IBMBibRecord is the IBM-specific version of the base BibRecord element.

Parts assemblies Parts assemblies are a special case of modular information for describing the parts of hardware products.

New modules may be added to the IBM ID Doc application at any time. In addition, third parties are free to define new modules by defining them using the module declaration processing instructions. One of the chief intents of making IBM ID Doc modular is to enable and encourage third party vendors or other software producers to extend the IBM ID Doc application in new and interesting ways.

In addition to the various modules, IBM ID Doc defines the following processing functions as being optional:

- Content reference links
- Retrieval alternative links
- Cross references outside the local document.
- Any HyTime features not required by the base IBM ID Doc module's HyTime support.

Content reference links allow the use of an element's content by reference from another element of the same type. Retrieval alternatives allow the “chaining” of elements of the same type off of a root element for the purpose of providing different versions of the same element, such as to provide different language versions of the same logical element. Cross references outside the local document provide a convenient way to refer to divisions or other elements in other documents without the overhead of an explicit hyperlink. Because these features can be expensive to implement, both in terms of program code and system resources needed during processing, these functions are optional.

It is also permissible for editors that implement direct support for IBM ID Doc elements and processing functions to not provide direct support for these optional features even though associated processors do implement them, since it always possible for editors to create the hyperlinks directly by manually specifying the necessary element IDs. However, the documentation for any editor that claims direct support for IBM ID Doc (as opposed to simply being able to support the DTD) must clearly indicate which optional features of IBM ID Doc it does not support. For an editor to claim support for IBM ID Doc, it must provide direct support for all IBM ID Doc-defined required elements and processing functions, over and above those functions inherent in SGML support in general. For example, an editor must

provide direct support for creating and traversing all non-optional hyperlinks defined by IBM ID Doc.

IBM ID Doc Base Element Set

The following elements, and any unique elements they may contain, make up the base set of elements of the IBM ID Doc application:

- Abbreviations
- Abstract
- Address
- AnchorRoleDef
- Annot
- APL
- Appendx
- Author
- Backmatter
- Bibliography
- BibList
- BibRecord
- Bin
- Body
- Bridge
- Caption
- Caution
- Char
- Citation
- Danger
- Dataloc
- Date
- Decimal
- DefinedTerm
- Desc
- Div
- DL
- FigureList
- Figure
- FN
- Frontmatter
- GL
- Glossary
- Hex
- I1
- I2
- I3
- Index
- InfoModule
- Iref
- LibRecord
- Lines
- Link
- LinkDesc
- Litdata
- LongQuote
- MD
- ModularInfo
- MV
- Name
- MultiMediaObject
- Note
- NoteList
- Notloc
- Number
- ObjectContainer
- Octal
- OL
- P (paragraph)
- Part
- Phrase
- Preface
- Prolog
- Query
- Quote
- RefKey
- Safety
- Screen
- Soa
- Title
- TableList
- TOC
- UL
- Warning
- XMP
- XPH
- XRef

Chapter 7. IBM ID Doc Elements

This section defines all the elements and attributes in the IBM ID Doc language.

Abbreviations . . . 38
Abstract . . . 39
Address . . . 40
AnchorRoleDef (hyperlink anchor role defininition) . . . 45
Annot (annotation) . . . 46
Annotation Anchor Role (annotation link) . . . 48
APL (APL data) . . . 49
Appendix . . . 50
Author . . . 51
Authors . . . 52
Backmatter . . . 53
Bibliography . . . 55
BibList (bibliography entry list) . . . 56
BibRecord (bibliographic record) . . . 58
Bin (binary or Boolean data) . . . 62
Body (document body) . . . 63
Bridge (bridge between concepts) . . . 64
Caption . . . 67
Caution (caution notice) . . . 67
CGraphic (character graphic) . . . 69
Char (character data) . . . 70
Citation (document citation) . . . 71
Class Attribute (element class) . . . 73
ClassDef (element class definition) . . . 74
Common Attributes . . . 77
ControlInfo (process-related control information) . . . 81
CopyRight (copyright attribution) . . . 85
CopyRights (copyright attributions) . . . 88
Danger (danger notice) . . . 89
Date . . . 90
Decimal (decimal number) . . . 91
DecisionPoint . . . 92
DefinedTerm . . . 93
Delim (syntactic delimiter) . . . 94
Desc (element's description) . . . 95
DescribedDocument Anchor Role (link to a described document) . . . 96
Div (hierarchical division) . . . 97
DivBody (division body) . . . 99
DivIntro (division introduction) . . . 100
DivProlog (division metainformation) . . . 104
DivSummary (division summary) . . . 106
DL (definition list) . . . 107
DLBlock (definition list block) . . . 108
DLEntry (definition list entry) . . . 109
DocTitle (document title) . . . 111
DocTrackingInfo (document tracking and control information) . . . 113
EditionNotices . . . 115
Enterprise (enterprise name and address) . . . 116
EnterpriseName (enterprise name) . . . 117

© Copyright IBM Corp. 1992, 1993

FigSeg (figure segment) . . . 118
Figure (exhibits) . . . 119
FigureList (list of figures) . . . 121
FN (footnote) . . . 122
Fragment (syntax fragment) . . . 124
FragmentRef (syntax fragment reference) . . . 125
Frontmatter . . . 127
GL (glossary list) . . . 129
GLBlock (glossary list block) . . . 131
GLEntry (glossary list entry) . . . 132
Glossary . . . 134
Group (syntax grouping) . . . 135
Hex (hexadecimal number) . . . 138
Hyperlink Attributes (hyperlink definitions) . . . 140
IBMBibRecord (IBM bibliographic record) . . . 141
IBMIDDOC (IBM-specific product documentation) . . . 145
IBMLibRecord (IBM document library definition) . . . 147
IBMSafety (IBM safety notices) . . . 150
ID Attribute (element identifier) . . . 152
Index . . . 152
InfoModule (information module) . . . 154
IRef (index entry reference) . . . 156
I1 (first-level index entry) . . . 157
I2 (second-level index entry) . . . 159
I3 (third-level index entry) . . . 161
Kwd (syntactic keyword) . . . 163
LE (language element) . . . 164
LEDesc (language element description) . . . 165
LEDI (language element description item) . . . 166
Legend . . . 169
LEName (language element name) . . . 171
LERS (language element reference section) . . . 172
LERSDef (LERS property definition) . . . 173
LI (list item) . . . 175
LIBlock (list item block) . . . 176
LibRecord (document library definition) . . . 177
Lines (text with significant line boundaries) . . . 181
Link (explicit hyperlink) . . . 182
LinkDesc (linked location description) . . . 184
LinkEnds, AnchorRoles, and HyTime Attributes (hyperlink target reference attributes) . . . 187
Litdata (literal data) . . . 190
LongQuote (quotation that stands alone) . . . 192
ManPage (Unix manual page) . . . 193
Mark (marked note definition) . . . 195
MarkAction (marked note action definition) . . . 196
MarkClass (marked note class definition) . . . 197
MarkDesc (mark description) . . . 199
MarkList (marked note list) . . . 200
MarkNote (marked note) . . . 202
MD (deleted phrase) . . . 204
Message (message or code description) . . . 205
MessageItem (message description item) . . . 206
MessageList List of message descriptions . . . 209

ModularInfoDef (modular information property definition) 210
ModularInformation 212
ModuleItem (module description item) 213
MultiMediaObject (non-text object) 215
MV (message variable) 217
Name (person's name) 218
Nameloc (named location) 220
NMList (list of IDs or entities) 222
NMQuery (named query) 225
Notation Attributes 227
Attributes for Graphic Notations 227
Note 230
NoteList (ordered note list) 231
Notloc (notation-specific location) 232
Number 233
ObjectContainer (data object container) 234
ObjectLibrary 236
Octal (octal number) 238
OL (ordered list) 239
Oper (syntactic operator) 240
P (paragraph) 241
ParaBlock (paragraph block) 243
ParaLevelElements (paragraph-level elements) 245
Parm (parameter list entry) 247
ParmBlock (parameter list block) 248
ParmList (parameter list) 249
Part (major document part) 251
Person (person's name and address) 253
Phrase (one or more words) 254
PhraseLevelElements 256
Preface 257
Procedure 258
ProcEntry (procedure entry point) 260
ProcExit (procedure exit point) 262
ProcStep (procedure step) 263
Prolog (document metainformation) 266
Property Attributes (element properties) 268
PropertyDef (property set definition) 271
PropertyDefinitions 272
PropertyGroup 273
PropertySource Anchor Role (property reference link) 275
PublicIdentifier 276
Quote (quotation phrase) 276
Range (syntactic range) 278
RefKey (reference key hyperlink) 281
RepSep (repeat separator) 284
Alternate Anchor Role (retrieval alternative link) 286
Revision 287
RevisionHistory (revision tracking information) 288
RevisionSpecification Anchor Role (revision specification link) 289
Safety (safety notices) 290
Screen (computer screen or panel) 291
Set (syntactic set) 293
SOA (summary of amendments) 296

StepNotes . . . 297
Style Attribute (element style information or reference) . . . 298
Syntax . . . 301
SyntaxBlock . . . 304
SyntaxNote . . . 306
SyntaxPhrase (syntax phrase) . . . 307
Table . . . 308
TableList . . . 311
Title (element title) . . . 312
TOC (table of contents) . . . 314
UL (unordered list) . . . 316
Var (syntactic variable) . . . 317
Warning (warning notice) . . . 318
Web (hyperlink web) . . . 319
WebNode (hyperlink web node) . . . 321
WebSet (set of hyperlink webs) . . . 324
Xmp (example of computer input or output) . . . 325
XPH (example phrase) . . . 326
XRef (cross reference) . . . 327

Abbreviations

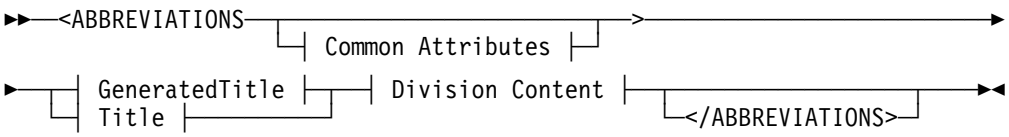
Purpose

Abbreviations is intended to contain an explanation of any abbreviations used in the document.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the Abbreviations division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

Division Content

Abbreviations has the same content rules as Div elements.

Usage

Use Abbreviations to contain lists of abbreviations used in the information. The content of Abbreviations can be anything, but the best way to create a list of abbreviations might be to define a class of Abbreviation for use on DefinedTerm elements and then use a generated glossary list to list those defined terms with a class of Abbreviation.

Examples

```
<frontmatter>
<abbreviations><generatedtitle>
<p>The following abbreviations are used in this document:
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  Abbreviations    - O (%SpecialDiv;) >

<!ATTLIST  Abbreviations
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

Abstract

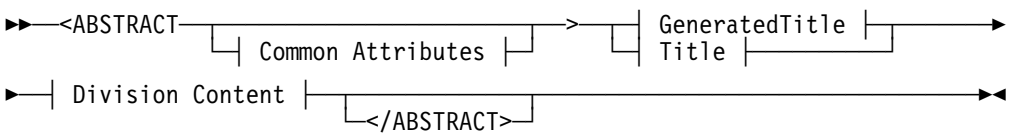
Purpose

Abstract is intended to contain a short description of the content of the document.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the Abstract division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

Division Content
Abstract has the same content rules as Div elements.

Usage

Use Abstract to contain a brief description of the document.

Examples

```
<frontmatter>
<abstract><generatedtitle>
<p>Discusses...
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  Abstract         - O (%SpecialDiv;) >
<!ATTLIST  Abstract
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

Address

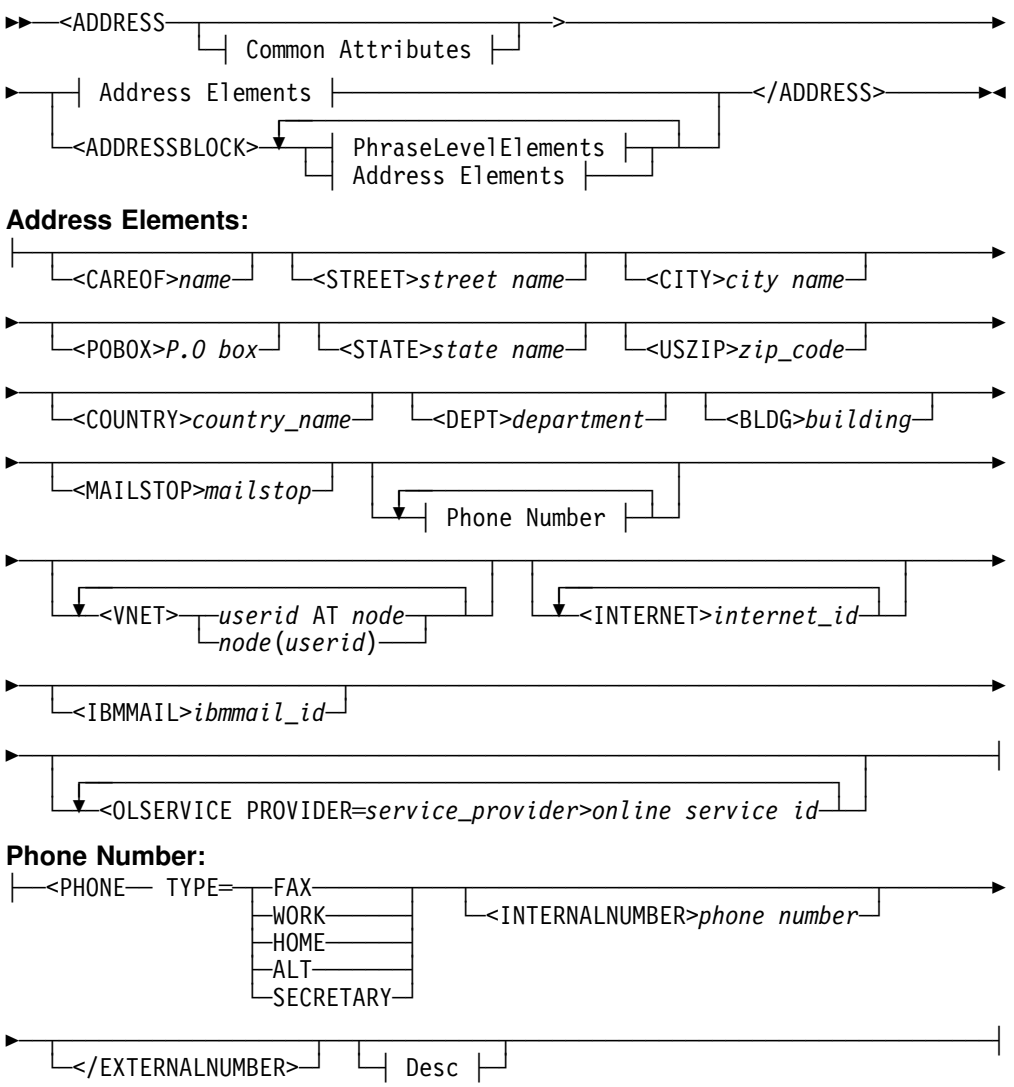
Purpose

Address contains the address of a person or enterprise. The address is structured to enable use of the address definition as a database.

Valid Contexts

Author, PhraseLevelElements.

Syntax



Attributes and Contained Elements

CAREOF
In care of information.

STREET
Street name.

POBOX
Post office box or similar item.

CITY
City name.

STATE
State or similar political entity (e.g., province).

COUNTRY
Country name.

USZIP
United States ZIP code, either a 5-digit number or a 9-digit number.

DEPT
Deparment identifier.

BLDG
Building identifier.

MAILSTOP
Mail stop, such as building and department.

PHONE
Phone number.

TYPE=
Phone number type, e.g. “Home”.

INTERNALNUMBER
The internal version of a number, for example, a TIE line or extension.

EXTERNALNUMBER
The external version of a number.

VNET
A VNET address. either in user AT node or node(user) format.

INTERNET
An internet address, e.g., drmacro@ralvm13.vnet.ibm.com.

IBMMAIL
An IBMMAIL ID, e.g., IBMDK934 at IBMMAIL.

OLSERVICE
Any online service address,

PROVIDER=*service provider*
The name of the service provider, e.g., Prodigy™.

Usage

Use the Address element to structure internal and external addresses. The Address element can take an ID for use by reference. Address is normally used within the context of an Author element but may be used elsewhere. The address elements must be specified in the order shown in the syntax definition.

Examples

```
<address>
<street>Cornwallis Rd.
<city>Research Triangle Park
<state>NC<uszip>27709
<dept>E14<bldg>500<MailStop>E14/B500
<phone  type=office>
<InternalPhone>Tie: 444-5160
<ExternalPhone>(919) 254-5160
<phone type=fax
<InternalPhone>Tie: 444-1234
<ExternalPhone>(919) 254-1234
<desc>JJ-109/B500
```

```
<vnet>DRMACRO at RALVM13
<internet>drmacro@ralvm13.vnet.ibm.com
<ibmmail>IBMDK934 at IBMMAIL
<olservice provider="Prodigy">adfadfds
</address>
```

SGML Definition

```
<!ELEMENT  Address          - -
                       (AddressBlock? |
                        (EnterpriseName?,
                         CareOf?, Street?, POBox?,
                         City?, State?, USZip?,
                         Country?,
                         Dept?, Bldg?, MailStop?,
                         Phone*,
                         VNET*, Internet*, IBMMail?, OLService*
                       ))
>
<!ATTLIST  Address
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Address"
>

<!ELEMENT  AddressBlock     - -
                       (%DataPool; | EnterpriseName |
                        CareOf | Street | POBox | City | State | USZip |
                        Country | Dept | Bldg | MailStop | VNET |
                        Internet | IBMMail | OLService | Phone)*
                                        -- Non-standard address block --
                       -- Use address subelements to do content markup. --
>
<!ATTLIST  AddressBlock
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "AddressBlock"
    Notation         (LineSpecific)
                                  LineSpecific
                         -- LineSpecific unless Object attribute used --
>

<!ELEMENT  (CareOf | Street | POBox | City | State | USZip |
            Country | Dept | Bldg | MailStop | VNET | Internet |
            IBMMail | OLService)
                                - O (%DataPool;)*
>

<!ATTLIST  (CareOf | Street | POBox | City | State | Country |
            Dept | Bldg | MailStop | IBMMail)
    %Common.Atts;
>

<!ELEMENT  Phone            - O (%DataPool;) >

<!ATTLIST  Phone
```

Address

```
                    Location         (Work | AltWork | Mobile | PDA | Home | AltHome |
                                      OtherLoc)   Work
                    Equipment        (Voice | VoiceMail | Fax | VoiceFax | VoiceMailFax |
                                      Pager | Modem | Telex | OtherEquip)
                                                  VoiceMail
                    Coverage         (Primary | Group | Secretary | AnsweringService |
                                      BBS | OtherCoverage)
                                                  Primary
                    Scope            (LocalPrivateNetwork | PrivateNetwork |
                                      IBMTieLine | InState | Domestic | International |
                                      OtherScope)
                                                  Domestic
                    Use              (BusinessHours | AfterHours | AllHours |
                                      Emergency | NoAnswer | OtherUse)
                                                  AllHours
                    OtherInformation CDATA        #IMPLIED
                    %Common.Atts;
                >

                <!ATTLIST  USZip
                    %Common.Atts.NoLexMdl;

                    LexModel         CDATA        #FIXED
                                          "#CONTENT ([0-9],[0-9],[0-9],[0-9],[0-9],
                                                     ('-',[0-9],[0-9],[0-9],[0-9])?
                                                    )
                                           &Common.LexModel;"
                                         -- HyLex restriction: 7- or 9-digit ZIP code --
                >

                <!ATTLIST  VNET
                    %Common.Atts.NoLexMdl;

                    LexModel         CDATA        #FIXED
                                          '#CONTENT ((name, S, "at", S, name) |
                                                                      (name, "(",name,")"))
                                           &Common.LexModel;'
                                         -- HyLex restriction: user at node or node(user) --
                >

                <!ATTLIST  Internet
                    %Common.Atts.NoLexMdl;

                    LexModel         CDATA        #FIXED
                                          '#CONTENT (name, "@", name, (".",name)*)
                                           &Common.LexModel;'
                                              -- HyLex restriction: name@name(.name)* --
                >

                <!ATTLIST  OLService
                    OLServiceName    CDATA        #IMPLIED
                    %Common.Atts;
                >
```

AnchorRoleDef (hyperlink anchor role defininition)

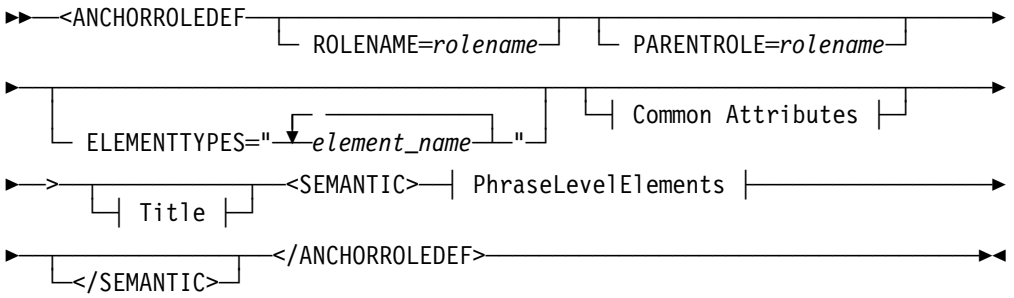
Purpose

AnchorRoleDef defines an anchor role name to be used on the common AnchorRole= attribute in conjunction with the common LinkEnds= attribute.

Valid Contexts

PropertyDefinitions.

Syntax



Attributes and Contained Elements

ROLENAME=*rolename*

The name of the anchor role being defined. Role names have the same scoping rules as element IDs, but are separate from element IDs.

PARENTROLE=*rolename*

The name of an anchor role that is the parent of this role. Using ParentRole=, you can define anchor role hierarchies.

ElementTypes=*element names*

Defines those element types (generic identifiers) to which this AnchorRoleDef applies. Use ElementTypes= when a AnchorRoleDef is only meaningful for a specific set of element types.

Title

An optional descriptive title. The title may be used, for example, in selection lists presented by authoring systems for selecting anchor roles.

Semantic

Defines the semantic meaning of a given role. Semantic is intended to document what a given anchor role means to the author that defined it.

Usage

Use AnchorRoleDef to define anchor roles that are specific to your information. Anchor roles let you assign meaning to specific hyperlink relationships. You can also use containment and element classes to imply hyperlink relationships. You may find that some relationships in your information are important enough to warrant both an element class and an anchor role.

You can associate hyperlink presentation and behavior styles or scripts, as well as other processing, with specific anchor roles. Do not define roles that are purely presentational, such as AutoLink or IconButton. Define classes that have a specific meaning.

Examples

```
<prolog>
  ⋮
<PropertyDefinitions>
 <AnchorRoleDef rolename=ExpertInfo>
  <title>Expert Information
  <semantic>Use for links from general information to
  information for experts.
 </AnchorRoleDef>
</PropertyDefinitions>
  ⋮
</prolog>
```

SGML Definition

```
<!ELEMENT  AnchorRoleDef    - - (Title?, Semantic)
                                                -- Class Definition --
>
<!ATTLIST  AnchorRoleDef
    RoleName         NAME         #REQUIRED
    ParentRole       NAME         #IMPLIED    -- Parent of this class --
    ElementTypes     NAMES        #IMPLIED -- GIs this Def applies to --
    %Common.Atts;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Content"
    InfoMaster       NAME         #FIXED    "PropertyDefiniton"
>
```

Annot (annotation)

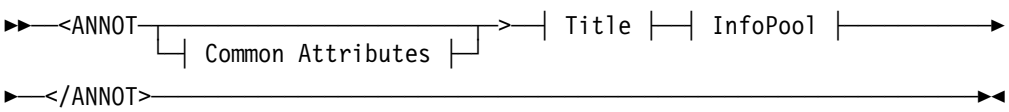
Purpose

Annot is used to annotate the content of another element.

Valid Contexts

InfoPool.

Syntax



Attributes and Contained Elements

Title

The title of the annotation. In some presentation styles, such as labeled boxes or pop-up windows, the title will be used as the title of the box or window.

InfoPool

Annot can contain both DataPool elements (including plain text) and InfoUnitPool elements.

Usage

Use Annot to contain comments about the content of an element, such as notes to reviewers, notes to editors, and so on. These annotations may be presented or suppressed, depending on the options given to the processing system. Whether they are presented or not, Annot's content must be valid (it will always be parsed).

A particularly powerful use of Annot is to define different classes of annotation and then use conditional processing based on the class value to present different annotations for different purposes. For example, in the description of a task, you might want to have one set of annotations for new users and another set of annotations for experienced users. You could define two classes of annotation, say NewUserNotes and ExperiencedUserNotes, and then use those classes on the Annot element, like so:

```
<p>To accomplish the framitizing task, do the following:
<annot class="NewUserNotes">Hints on Framitizing Success
<p>Framitizing can be difficult if you are not
experienced with the use of Wangdoozlers.
Make sure you are confident of your skill with the
Wangdoozler before committing any data as a result
of this task.
</annot>
<annot class="ExperiencedUserNotes">Speeding the Framitizing Process
<p>You can speed the Framitizing process dramatically if
your data conforms to certain patterns.
See <xref refid=tuningFramitizing> for details on how to
structure your data to improve performance.
</annot>
```

If your online presentation system provides sufficient conditional processing function, you can define a presentation style such that either the new user note or the experienced user note will be presented depending on the reader's current skill level setting. You can use this technique to easily provide customized information for different purposes from a single information base.

Migration Note

Unlike DCF BookMaster's Annot, Annot must not span document structures but rather is an element in the document hierarchy, like any other element. Like FN, it annotates the content of its containing element. The use of Annot to “comment out” information is an improper use of the Annot element.

The default presentation of an annotation element is within a labeled box.

Annot is valid anywhere text is valid, for example, within paragraphs, list items, and so on.

Examples

Including an annotation within text:

```
<p>This text is a paragraph before the annotation.</p>
<!-- notice the paragraph was ended so that the paragraphs would be
     peers, the annot would not be a child of the previous paragraph.-->
<annot>
<p>This is an annotation, the first paragraph.
<p>This is the second paragraph of the annotation
</annot>
<p>This is a paragraph after the annotation.
```

SGML Definition

```
<!ELEMENT  Annot            - - (Title, AnnotBody)      -- Annotation -->
<!ATTLIST  Annot
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!ELEMENT  AnnotBody        O O (%InfoPool;)*      -- Annotation Body -->
<!ATTLIST  AnnotBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ParagraphBody"
    -- Its semantic class is inherited from its parent: Annot --
>
```

Annotation Anchor Role (annotation link)

Purpose

An annotation hyperlink lets an element refer directly to elements that serve to annotate its content. An annotation link establishes an annotation relationship between two elements such that the element refered to is defined as annotating the element that refers to it. The expression of this relationship is a function of the active presentation style.

Usage

An annotation hyperlink lets you point from an element to another element that serves as an annotation for the element. For example, you may have a common safety notice that applies to a variety of elements throughout your document. You can specify the safety notice once and use the annotation anchor role to point to it from each element to which the notice applies.

As for all other ID references, an annotation hyperlink can refer to elements indirectly via a LinkDesc or Nameloc element.

Examples

```
<Warning id=datahazard><title>Hazard to Data
<p>Performing this function may harm data.
Ensure that proper backups have
been made first.
</Warning>
  ⋮
<p  linkends="datahazard" HyTime=ilink
 AnchorRoles="linksource annotation">
If you are feeling lucky, just turn
the machine off.
```

Note that the first role name, linksource, refers to the paragraph element itself, indicating that it is the source of the link.

APL (APL data)

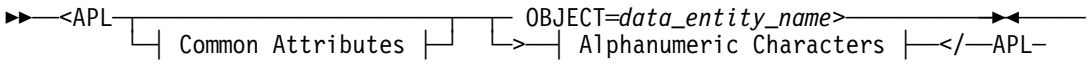
Purpose

The APL element identifies APL program fragments.

Valid Contexts

PhraseLevelElements.

Syntax



Attributes and Contained Elements

OBJECT=*data_entity_name*
The name of the data entity that contains the APL data. When OBJECT= is specified, it is an error to specify any data or the APL end tag.

Alphanumeric Characters
Any characters meaningful as APL data.

Usage

Use the APL element to identify data that is part of an APL program. The content of the element may be APL data or other elements from PhraseLevelElements that make sense in the APL context. This data is encoded in the document source using the character encoding used for APL data and programs, not necessarily the character encoding used for the data everywhere else in the document. The content of this element is typically presented using the same font as is conventionally used for APL, which will also probably differ from that used to present the other data found in the document. The content of this element can be created using an APL-capable terminal by changing the keyboard into APL mode and typing the data.

An external entity containing the APL data may be refered to using the Object= attribute, which must contain the name of a data entity.

Examples

```
A matrix is defined with the string
<APL>2 3⍴1 2 3 4 5 6</APL>
```

SGML Definition

```
<!ELEMENT  APL              - - (#PCDATA)          -- APL data phrase -->
<!ATTLIST  APL
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    Notation         (APL)
                                  #FIXED    "APL"
    ApplicationClass NAME         #FIXED    "APLData"
>
```

Appendix

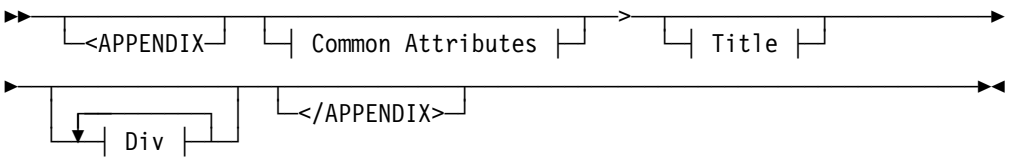
Purpose

Contains Div elements that are to be considered appendixes.

Valid Contexts

BackMatter.

Syntax



Attributes and Contained Elements

Title
The appendix division title.

Div
Appendix can contain only divisions.

Usage

Use Appendix to contain the appendixes for your document.

In the default presentation style, appendixes are numbered with letters rather than digits.

Appendix has an optional Title.

Examples

```
<appendix>
<div>First Appendix Title
</div>
<div>Second Appendix Title
</div>
</appendix>
```

SGML Definition

```
<!ENTITY % Appendix         "Title?, Div*" >

<!ELEMENT  Appendix         - O (%Appendix;) >
<!ATTLIST  Appendix
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

Author

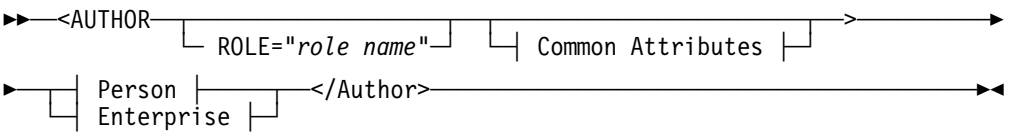
Purpose

Contains the name and address of an author.

Valid Contexts

Authors.

Syntax



Attributes and Contained Elements

ROLE=*role name*
Indicates the role the author played in the creation of the document, for example “Editor”.

Person
Contains the name and address of a person.

Enterprise
Contains the name and address of an enterprise.

Usage

Use Address to contain the name and address of a person or enterprise.

Examples

```
<author role="Editor">
 <person>
 <name id=drmacro><last>Kimber<first>W<middle>Eliot<honorific>Mr.</name>
 <address>
 </address>
 <person>
</author>
```

SGML Definition

```
<!ELEMENT  Author           O O (Enterprise | Person) >
<!ATTLIST  Author
    Role             CDATA        #IMPLIED           -- Author's role --
    %Common.Atts;
>
```

Authors

Purpose

Contains one or more authors.

Valid Contexts

BibRecord, IBMBibRecord.

Syntax

Attributes and Contained Elements

AUTHOR

One or more Author elements

Usage

Authors Organizes the Author elements with a BibRecord or IBMBibRecord element.

Examples

```
<authors>
 <author role="Editor">
  <person>
   <name id=drmacro><last>Kimber<first>W<middle>Eliot<honorific>Mr.</name>
   <address>
  </address>
  </person>
 </author>
</authors>
```

SGML Definition

```
<!ELEMENT  Authors          - - (Author)+ >
<!ATTLIST  Authors
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "Authors"
>
```

Backmatter

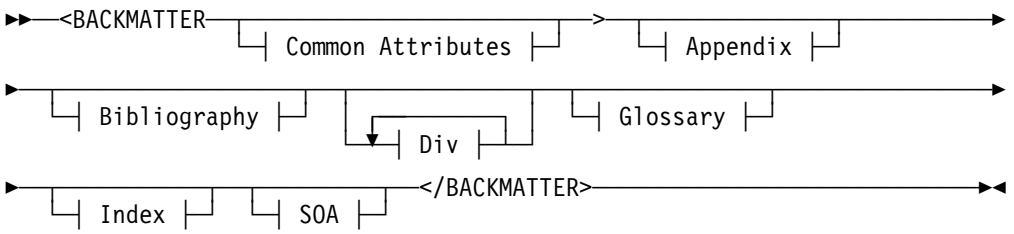
Purpose

Contains backmatter elements such as Glossary and Index.

Valid Contexts

IBMIDDoc, following Body

Syntax



Attributes and Contained Elements

Appendix
Contains any appendixes for the document.

Bibliography
Contains the document bibliography. The content of Bibliography is normally one or more BibList elements.

Glossary
Contains the document glossary The content of Glossary is normally one or more GL elements.

Index
Contains the index for the document, normally generated automatically.

SOA
Contains the document summary of amendments. The content of SOA is normally one or more MarkList elements.

Div
Division elements with any valid division content.

Usage

Use Backmatter to contain any backmatter elements for the document. The Backmatter specialized divisions are valid only within Backmatter, and possibly Frontmatter. Use Div elements to contain information not accounted for by the specialized Backmatter divisions.

Examples

```
<IBMIDDOC>
<Prolog>
  ⋮
</Prolog>
<Frontmatter>
  ⋮
</Frontmatter>
<Body>
  ⋮
<Backmatter>
<Glossary>
<gl>
</gl>
<Bibliography>
<BibList>
</BibList>
<Index>
</Backmatter>
</IBMIDDOC>
```

SGML Definition

```
<!-- '|' should be '&' and only Div allowed multiple times
     but tools don't properly support this *yet* -->
<!ENTITY % Backmatter
    "(Appendix? | Bibliography? | Glossary? | Index? | SOA? | Div)*"
>

<!ELEMENT  Backmatter       - O (%Backmatter;) >
<!ATTLIST  Backmatter
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

Bibliography

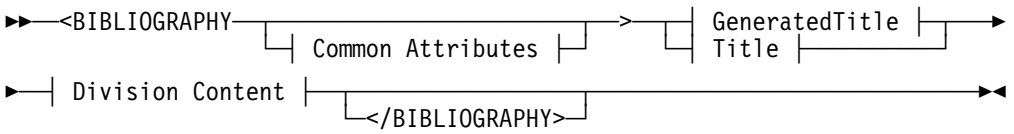
Purpose

Bibliography is intended to contain a list of documents or other materials refered to from or related to the current document.

Valid Contexts

Frontmatter or Backmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the Bibliography division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

Division Content

Bibliography has the same content rules as Div elements.

Usage

The bibliography division is intended to contain lists of documents and other materials relevant to the information. You can use BibList elements within Bibliography to contain or generate bibliography lists.

Examples

```
<backmatter>
<Bibliography><generatedtitle>
<p>See the following documents for more information:
<biblist>
  ⋮
</biblist>
</backmatter>
```

SGML Definition

```
<!ELEMENT  Bibliography     - O (%SpecialDiv;) >
<!ATTLIST  Bibliography
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

BibList (bibliography entry list)

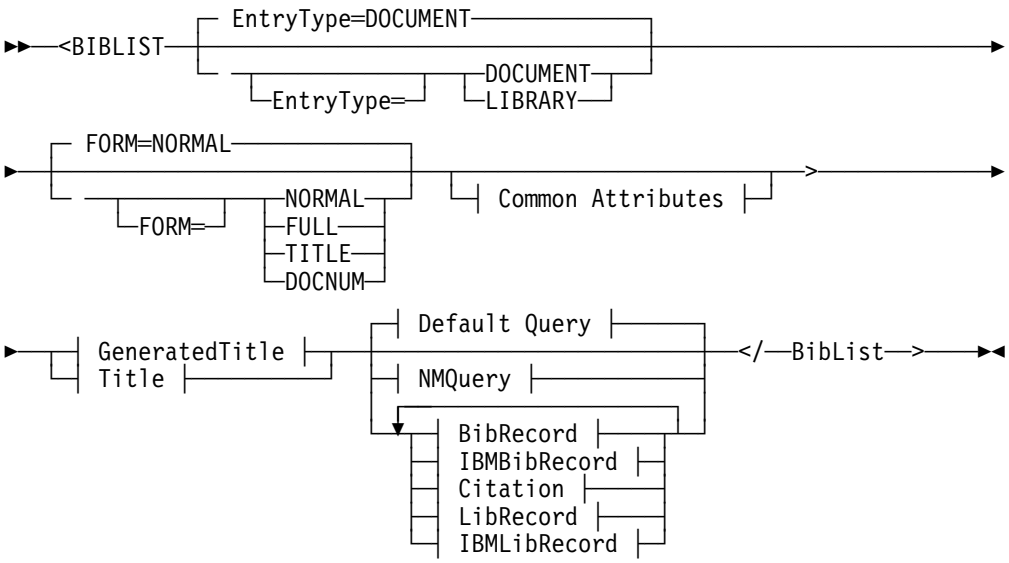
Purpose

BibList either generates a list of bibliography entries from the BibRecord and LibRecord elements used or refered to from within the information, or contains a set of BibRecord, LibRecord, or Citation elements that define the BibList content.

Valid Contexts

ParaLevelElements (normally used in Bibliography).

Syntax



Attributes and Contained Elements

ENTRYTYPE=DOCUMENT | LIBRARY
Indicates whether or not the generated list is to contain BibRecord entries or LibRecord entries.

FORM=*form value*
Defines the form of the generated bibliography entries. The specific meaning of **FULL** and **NORMAL** is defined by the active style.

GeneratedTitle
Title
You must specify either GeneratedTitle or Title for the BibList element. If you specify Title the title start tag must be specified.

Default Query
If neither NMQuery nor any contained elements are specified, then the default query (as defined by the active style definition) will be used to generated the content of the BibList element. The recommended default query is to list all Bibrecord elements in the document, sorted alphabetically by author or title (if there is no author).

NMQuery
Contains the query that defines the content of the generated bibliography list. The content of the NMQuery may be a reference to a pre-defined query. A typical query for use within a Biblist element would be “Find all BibRecord and IBMBibrecord elements and sort them alphabetically by author's last name.”

Pre-defined queries are provided by a particular IBM ID Doc implementation.

BibRecord
IBMBibRecord
Citation
LibRecord
IBMLibRecord
You can create an explicit bibliography list by building it out of any combination of bibligraphy records, library records, or Citation elements.

Usage

Use BibList to either contain a set of BibRecord and LibRecord elements, or a query that will generate a list bibliography entries at any point in the information.

BibList is normally used within a Bibliography division.

Examples

```
<Bibliography>
<!-- Generate bibliography from all BibRecords used -->
<biblist form=full><generatedtitle>
<!-- Note lack of any content causes default query to be used -->
</biblist>
```

Example of BibList used to contain set of citations to BibRecords:

```
<Bibliography>
<biblist form=full><title>IBM ID Doc Library
<citation bibid=sbkmbookieug>
<citation bibid=sbkmbookieref>
</biblist>
```

Example of BibList used generate a bibliography from a LibRecord

```
<Bibliography>
<biblist form=full entrytype=library>
<citation bibid=sbkmbookielib>
</biblist>
```

Example of BibList containing BibRecords directly:

```
<Bibliography>
<biblist form=full>
 <bibrecord><doctitle><topic>The Whole Internet Catalog</tocic>

  ⋮
 </bibrecord>
 <bibrecord><doctitle><topic>The Hitchhiker's Guide to the Internet
   </topic>

  ⋮
```

```
 </bibrecord>
</biblist>
```

SGML Definition

```
<!ELEMENT  BibList          - O (Title?,
                                 (nmquery |
                                  (%BibRecord; | Citation | %LibRecord;)*
                                ))
                                                  -- Bibligraphy List --
>
<!ATTLIST  BibList
    EntryType        (Document | Library)
                                  Document
    Form             (Full | Title | TitleNumber | Normal)
                                  Normal
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleList"
>
```

BibRecord (bibliographic record)

Purpose

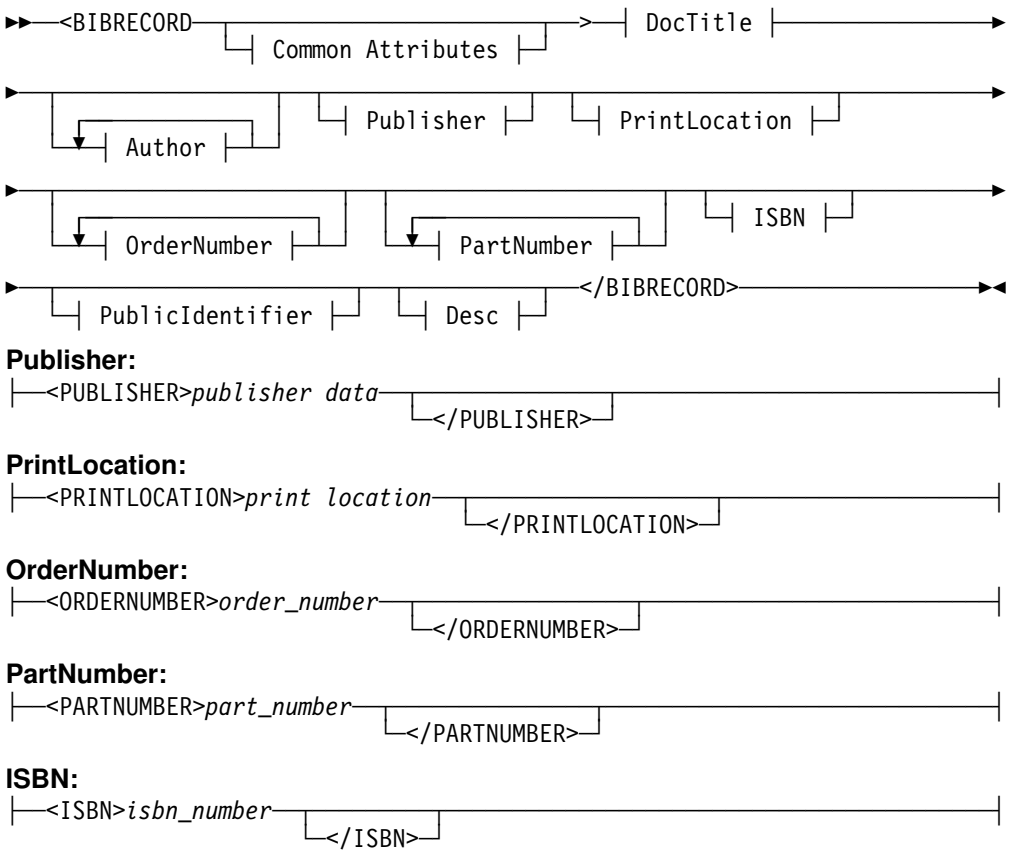
Bibliographic records structure and organize the information about documents. The bibliographic information can be used in a variety of ways, including generating bibliographies.

Use IBMBibRecord to define bibliographic entries for IBM documents.

Valid Contexts

BibRecords, BibList, Citation.

Syntax



Attributes and Contained Elements

DocTitle
The title of the document.

Author
One author of the document. You can specify multiple authors.

Publisher
Contains the name of the publisher followed by an optional address element.

PrintLocation
The country where the document is printed.

OrderNumber
The order number assigned to this document.

ISBN
The ISBN number assigned to this document.

PublicIdentifier
The SGML public identifier assigned to this document. This is the same public identifier used in entity declarations for this document.

Desc
A Desc element containing a description of the document.

Usage

Use BibRecord to define the bibliographic information about a document. Use IBMBibRecord to define bibliographic information for IBM documents.

BibRecord elements can also be specified in a BibRecords element or object container and used by reference from within a document, for example, from Citation and Quote elements.

When Linkends= is used with the anchor role DescribedDocument, the BibRecord element becomes the anchor of a link to the document described. When the unified name space option is in effect, the Linkends= value can be the name of the entity that represents the described document. When the unified name space option is not in effect, you must create Nameloc element to map the ID reference used in the Linkends= value to the entity name (see “Nameloc (named location)” on page 220 for details on using the Nameloc element, and “LinkEnds, AnchorRoles, and HyTime Attributes (hyperlink target reference attributes)” on page 187 for details on using the LinkEnds= attribute).

The default hyperlink style is for the BibRecord and all references to it to be active hyperlinks to the document.

When BibRecord is specified within Citation, Quote, and LongQuote, the BibRecords are collected for use in a generated bibliography (see “BibList (bibliography entry list)” on page 56).

Examples

```
<bibrecord id=bookieref>
 <DocTitle>
 <shorttitle>BookMaster Reference
 <Library>IBM ID Doc
 <topic>Language Reference
 </DocTitle>
 <author><name>Day<first>Don<middle>R.</name>
 </author>
 <author><name>Kimber<first>William<middle>E.</name>
 </author>
 <author><name>Temple<first>Mike</name>
 </author>
 <author><name>Wohler<first>Wayne<middle>L.</name>
 </author>
 <Publisher>TIC Corporation
  <address>
  </address>
 </publisher>
 <PrintLocation>USA
 <OrderNumber>SC32-5009-11
 <PublicIdentifier>+//ISBN 0-19-9999
 //PUB SC32-5009-11
 /IBM ID Doc Reference
 /drmacro@ralvm13.vnet.ibm.com
 /Publishing Solutions
 //EN
 <Desc>Defines the BookMaster language.
 BookMaster is an SGML language for creating printed and online
```

```
 technical information.
</bibrecord>
```

SGML Definition

```
<!ELEMENT  BibRecord        - O
                     (DocTitle,
                      Authors?,
                      Publisher?, PrintLocation?,
                      OrderNumber?,
                      ISBN?, PublicIdentifier?, Desc?)
                                          -- Bibliographic Record --
>
<!ATTLIST  BibRecord
    %LibRecord.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
    ApplicationClass NAME         #FIXED    "BibliographicRecord"
>

<!ELEMENT  Publisher        - O (EnterpriseName, Address?) >
<!ATTLIST  Publisher
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
             -- This element also conforms to the NameAndAddress form --
    ApplicationClass NAME         #FIXED    "Publisher"
>

<!ELEMENT  PrintLocation    - O (%DataPool;)* >
<!ATTLIST  PrintLocation
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "PrintLocation"
>

<!ELEMENT  OrderNumber      - O (#PCDATA)    -- Customer Order Number -->
<!ATTLIST  OrderNumber
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "OrderNumber"
>

<!ELEMENT  ISBN             - O (#PCDATA) >
<!ATTLIST  ISBN
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ISBN"
    LexModel         CDATA        #FIXED
       '(digit, "-", digit, digit, "-", digit, digit, digit, digit,
         digit, digit, ("-", digit)?)
        &Common.LexModel;'
>
```

```
<!ELEMENT  PublicIdentifier - O (#PCDATA) -- Formal Public Identifier -->
<!ATTLIST  PublicIdentifier
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "PublicIdentifier"
    LexModel         CDATA        #FIXED
       '(("-" | "+"), "//", owneridentifier, "//", publictextidentifier,
        "//", languagecode)
        &Common.LexModel;'
>
```

Bin (binary or Boolean data)

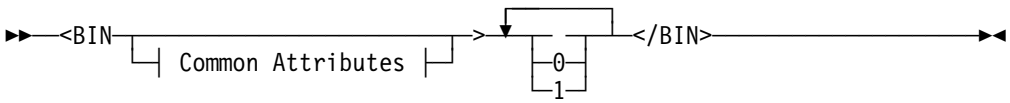
Purpose

The Bin element identifies binary data.

Valid Contexts

PhraseLevelElements.

Syntax



Usage

Use the Bin element to identify binary data. Its content is a space or the two binary digits, '0' and '1', with '0' representing the “off” value and '1' representing the “on” value. A space character in the content of this number represents a visual separator within a single binary number (not a separator between multiple binary numbers).

Examples

```
<BIN>11000001</BIN>
```

SGML Definition

```
<!ELEMENT  Bin              - - (#PCDATA)       -- Binary data phrase -->
<!ATTLIST  Bin
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "BinaryData"
    LexModel         CDATA        #FIXED
                          "#CONTENT ('0' | '1' | ' ')*
                           &Common.LexModel;"
>
```

Body (document body)

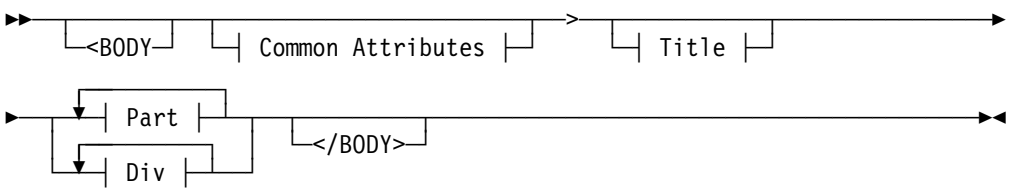
Purpose

Contains the document body.

Valid Contexts

Part, Div.

Syntax



Attributes and Contained Elements

Title
An optional title for the body.

Part
Div
Body can contain only divisions or parts.

Usage

Use Body to contain the main body of your document.

The Body start tag is omissible, as is the IBMIDDOC start tag. Thus, the minimum valid IBM ID Doc document consists of a single division and its title, followed by any valid division body element or a DivBody tag:

```
<div>Division Title
<p>Paragraph.
</div>
```

or:

```
<div>Division Title
<DivBody>
</div>
```

You can take advantage of this aspect of IBM ID Doc to process any division independently of the rest of your document. You need only ensure that the required entity declarations are available when you process the document. How this is accomplished is a function of the processing system you are using and is not defined by IBM ID Doc

Examples

```
<body>
<div>First Section Title
</div>
<div>Second Section Title
</div>
</body>
```

SGML Definition

```
<!ENTITY % Body             "Title?, (Part* | (Div | Procedure)*)" >

<!ELEMENT  Body             O O (%Body;) >
<!ATTLIST  Body
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

Bridge (bridge between concepts)

Purpose

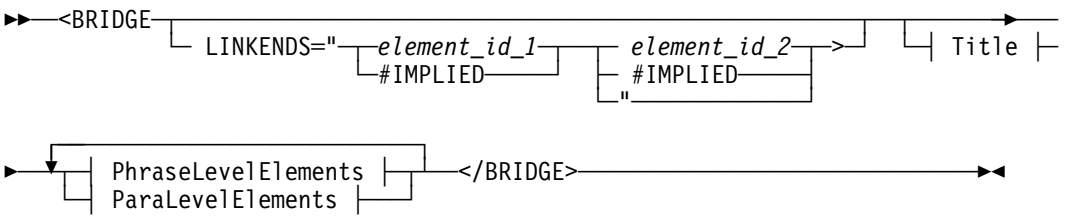
The Bridge element defines a hyperlink between two elements that embodies the semantic of bridging from one concept to the other. The simplest expression of this hyperlink is as a title with no hierarchical signficance.

The Bridge element may contain information that explains or provides the details of why the two concepts have been bridged.

Valid Contexts

ParaLevelElements, DL, DLBlock, GL, NoteList, OL, ParmList, StepNotes, UL.

Syntax



Attributes and Contained Elements

LINKENDS="*element ids***"**

The *element ids* are optional identifiers of the element locations that are to be linked or bridged. If the IDs are not specified, the element to be linked is defaulted. For *element_id_1*, the default is the element preceding the Bridge element. For *element_id)2*, the default is the element following the Bridge element. If you want to specify only one of the link ends explicitly, you must specify the keyword **#IMPLIED** for the other element.

Usage

Use the Bridge element to link two locations in a document together, providing the text that explains the linkage. A good example of a bridge is in the magazine publishing world. Bridges would be used to identify the titles that are sprinkled throughout magazine articles to bridge the reader from one topic to the next, when the article is making such a transition. The title does not define a hierarchical division in an article, but serves merely to provide a transition from one part to the next.

The simplest use of Bridge is to create just such a “bridging title” by specifying a title for the Bridge element and no LinkEnds= value.

By using the LinkEnds= attribute, you can use Bridge to connect any two elements that your system is capable of linking to, enabling the creation of very sophisticated hyperlink structures. For example, in an online presentation system, you may want to define navigation aids by using Bridge elements to contain the explanation of why certain concepts are linked together. For example, you could use Bridge elements to link tasks descriptions and function descriptions together, with the Bridge element containing the explanation of how the task relates to the function.

The uses to which Bridge can be put is limited only by the hyperlinking and presentation functions of your system.

Examples

Bridging sequencial elements:

```
<p>In summary, widget X is the price performance leader in its class.
It ...
⋮
<Bridge>For even more power for a very small investment
<p>Widget Y is also the best price performance leader in its class
...
```

Bridging non-sequencial elements:

```
<p>In summary, widget X is the price performance leader in its class.
It ...
<bridge refids='#implied widgety">
For even more power for a very small investment
</bridge>
⋮
<p id=widgety>
Widget Y is also the best price performance leader in its
class ...
```

Bridge used a list to connect to sets of list items:

Bridge (bridge between concepts)

```
<ul>
<liblock>
<li>Apples
<li>Bannanas
</liblock>
<bridge>
<p>the entries for C through X have been
omitted for simplicity.
</bridge>
<liblock>
<li>Yumpkins
<li>Zagmots
</ul>
```

The next example shows Bridge used in a separate document to form a navigation aid between a user's guide and a reference document. In this example, all the link targets are defined elsewhere using LinkDesc elements, not shown in this example.

```
<div>How Tasks Relate to Function
<p>This section describes how each task
described in <citation bibid=usersguide>
relates to the product functions
defined in <citation bibid=refmanual>.
<bridge linkends="task1 funca"><title>Task One
<p>Task one involves the use of Funcation A to
enable the creation framitz wozzlers.
</bridge>
  ⋮
<bridge linkends="taskn funcz"><title>Task The Last
<p>The last task
destroys any framitz wozzlers you may have created as a
result of performing task one.
</bridge>
</div>
```

SGML Definition

```
<!ELEMENT  Bridge           - O (Title?, (%InfoUnitPool;)*) >
<!ATTLIST  Bridge
--  LinkEnds         NAMES        #IMPLIED  part of common attributes --
    %Common.Atts.NoLexMdl.NoHyTime;

    InfoMaster       NAME         #FIXED    "ConceptBridge"
    HyTime           NAME         #FIXED    "ilink"
    anchrole         NAMES        #FIXED    "link1 link2"
    LexModel         CDATA        #FIXED
                          'LinkEnds ((name | "#IMPLIED"), S,
                                          (name | "#IMPLIED")
                           &Common.LexModel;'
>
```

Caption

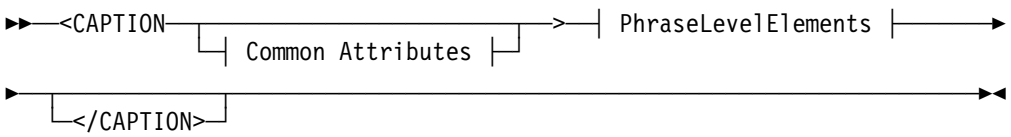
Purpose

Caption contains a caption for a display element.

Valid Contexts

Figure.

Syntax



Attributes and Contained Elements

PhraseLevelElements

Caption can contain any phrase-level elements.

Usage

Use caption for figures and tables. The caption is the text normally presented in lists and cross references and is intended to be a relatively short description of the element.

Examples

```
<figure id=figa>Title of Figure
<caption>This is the figure caption
<desc>This is the figure description.
<figseg>
</figseg>
</figure>
```

SGML Definition

```
<!ELEMENT  Caption          - O (%DataPool;)* >
<!ATTLIST  Caption
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Caption"
>
```

Caution (caution notice)

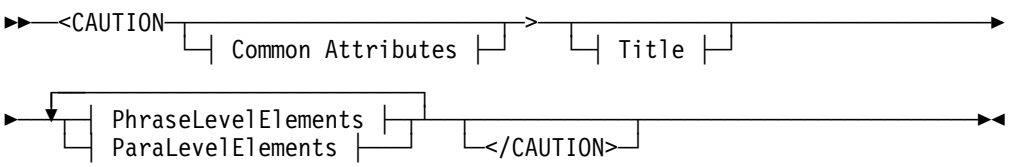
Purpose

Contains a caution notice.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

Title
In the default style, the generated English language title for Caution is 'Caution:'. When specified, the Title replaces any generated title.

PhraseLevelElements
ParaLevelElements
Caution has the same content model as paragraphs, except that Caution can contain paragraphs

Usage

Use Caution to create a caution notice, consisting of one or more paragraphs or other paragraph-level elements. Cautions are normally used to warn about actions that may cause damage to equipment.

Examples

Simple example:

```
<caution>Running your engine without oil may cause irreparable damage.
```

Example with title:

```
<Caution><title>Caution title</title>
Caution content
```

SGML Definition

```
<!ELEMENT  Caution          - O (Title?, NoteBody) >
<!ATTLIST  Caution
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>
```

CGraphic (character graphic)

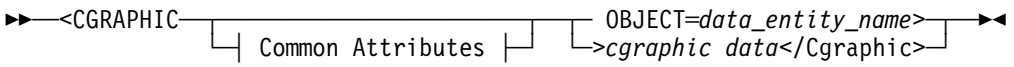
Purpose

Contains a graphic created with box and line characters.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

OBJECT=*data_entity_name*
Refers to the SGML data entity (CDATA, or NDATA) that contains the character graphic. When Object= is specified, it is an error to specify the CGraphic end tag.

character graphic data
The data making up the graphic.

Usage

Use CGraphic with Figure and Screen to create simple box and line graphics. The actual characters used to create a character graphic are defined by your processing system.

Migration Note

DCF BookMaster CGraphics can contain characters that are not part of the IBM ID Doc document character set and thus **must** be made into external NDATA entities and refered to using the Object= attribute. When migrating existing DCF BookMaster documents, the migration tool will do this for you automatically.

Examples

```
<figure><Caption>Simple box and line graphic
<CGraphic object=mygraphic>
</figure>
```

SGML Definition

```
<!ELEMENT  CGraphic         - O RCDATA           -- Character graphic -->
<!ATTLIST  CGraphic
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    Notation         (CGraphic)
                                  #FIXED    "CGraphic"
    ApplicationClass NAME         #FIXED    "CharacterGraphic"
>
```

Char (character data)

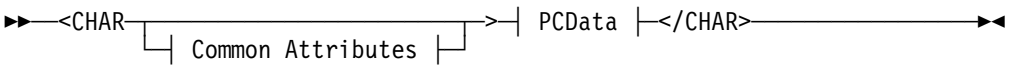
Purpose

The Char element identifies literal character data.

Valid Contexts

PhraseLevelElements

Syntax



Usage

Use the Char element to identify literal character data. Its content is any graphic character (as opposed to control characters) in the document character set or any defined character entity. The content may also contain space characters.

Examples

```
<char>This is a literal character string</char>
```

SGML Definition

```
<!ELEMENT  Char             - - (#PCDATA)
                                    -- literal character data phrase --
>
<!ATTLIST  Char
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "CharacterData"
>
```

Citation (document citation)

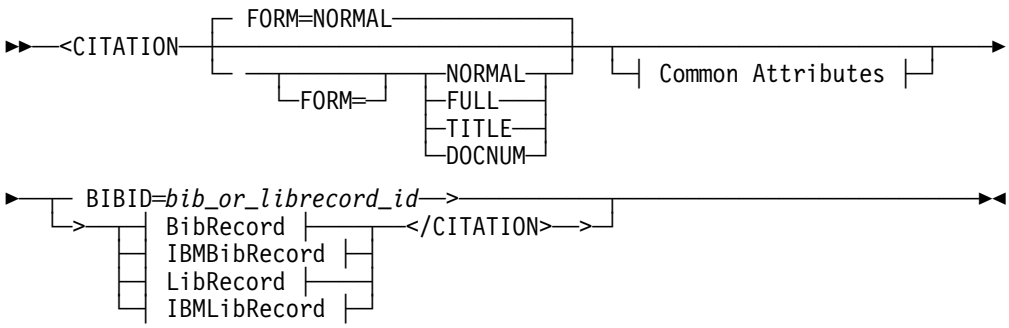
Purpose

Contains a citation to another document. When the BibRecord for the cited document defines an active hyperlink, the citation element becomes an active hyperlink to the document as well.

Valid Contexts

PhraseLevelElements, BibRecord, BibList.

Syntax



Attributes and Contained Elements

BIBID=*bib_or_librecordid*

Refers to a BibRecord or LibRecord element defined elsewhere. When BibID= is specified, it is an error to specify any content or the Citation end tag.

BibRecord

A BibRecord element can be contained within a Citation element. The default presentation is for the document title to be presented at the point the Citation is specified. The BibRecord will be collected for use in a generated bibliography list (see “BibList (bibliography entry list)” on page 56).

LibRecord

A LibRecord element can be contained within a Citation element. The default presentation is for the library title to be presented at the point the Citation is specified. The LibRecord will be collected for use in a generated bibliography list (see “BibList (bibliography entry list)” on page 56).

Usage

Use Citation to create a reference to a document within paragraph-level element content.

Citation can be used one of two ways. It can either refer to a BibRecord or LibRecord defined elsewhere (for example, in a BibRecords element in a division prolog), or it can directly contain a BibRecord or LibRecord element. When Citation refers to a BibRecord or LibRecord element, it is an error to specify the Citation end tag.

Citation (document citation)

When Citation contains a BibRecord or LibRecord, the default presentation is for the document title to be presented at that point in the information and for the BibRecord or LibRecord to be collected for later use in a generated bibliography list. Contained BibRecord elements and LibRecord elements are normally used within re-used information in order to ensure that the re-used information is completely self-contained.

It is expected that in normal production environments, sets of BibRecord and LibRecord definitions will be defined centrally and used by reference exclusively.

Use the Form= attribute to control the form of the citation. For example, if you only want the document number to be presented, you would specify `form=docnum`. The meaning of the Form value `'normal'` is defined in the active style.

Examples

```
<DivProlog>
  ⋮
<BibRecords>
<bibrecord id=bookieref>
 <DocTitle>
 <shorttitle>BookMaster Reference
 <Library>IBM ID Doc
 <topic>Language Reference
 </DocTitle>
 <author><name>Day<first>Don<middle>R.</name>
 </author>
 <author><name>Kimber<first>William<middle>E.</name>
 </author>
 <author><name>Temple<first>Mike</name>
 </author>
 <author><name>Wohler<first>Wayne<middle>L.</name>
 </author>
 <Publisher>IBM Corporation
  <address>
  </address>
 </publisher>
 <PrintLocation>USA
 <DocNumber>SC32-5009-11
 <PublicIdentifier>+//ISBN 0-19-9999
 //PUB SC32-5009-11
 /IBM ID Doc Reference
 /drmacro@ralvm13.vnet.ibm.com
 /Publishing Solutions
 //EN
 <Desc>Defines the BookMaster language.
 BookMaster is an SGML language for creating printed and online
 technical information.
</bibrecord>
</BibRecords>
</DivProlog>
  ⋮
<p>See <citation bibid=sgmlbookieref>
for more information.
```

Example showing contained BibRecord:

```
<p>To better understand the intracies of SGML, see
<citation>
<BibRecord>
<DocTitle>
<Topic>The SGML HandBook
<Author><name><last>Goldfarb<first>Charles<middle>F.
<Publisher>
Oxford University Press
Walton Street
Oxford OX2 6DP
</Publisher>
<PrintLocation>Printed and Bound in Great Britain
<ISBN>0-19-835737-9
<PublicIdentifier>+//ISBN 0-19-853737-9//DOCUMENT The SGML Handbook//EN
</BibRecord>
</Citation>
for more information.
```

SGML Definition

```
<!ELEMENT  Citation         - O (%BibRecord; | %LibRecord;) >
<!ATTLIST  Citation
    BibID            NAME         #CONREF             -- BibRecord ID --
    Form             (Full | Title | DocNum | Normal)
                                  Normal
    ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    %Common.Atts.Base;

    InfoMaster       NAME         #FIXED    "phrase"
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED
                          "BibID (&BibRecord;)
                           &Common.RefType;"
>
```

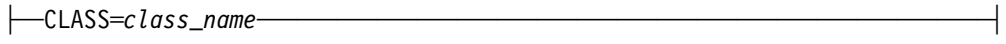
Class Attribute (element class)

Purpose

Associates an author-defined element class with an element. Author-defined element classes provide a way for authors to provide precise identification of the meaning of given elements beyond the identification provided by the element name itself. Element classes are defined with ClassDef elements within PropertyDefinitions.

Syntax

Class= Attribute:



Usage

Use author defined element classes to precisely describe document content. See “ClassDef (element class definition)” for more information on author-defined element classes.

Examples

```
<prolog>
  ⋮
<PropertyDefinitions>
 <PropertyDef version='v2r3'>
 <ClassDef ElementType=figure classname=codesamples>
  <title>Code Samples
  <IndexTerm>code sample
  <semantic>Figures that contain code samples.
 </ClassDef>
</PropertyDefinitions>
  ⋮
</prolog>
  ⋮
<figure class=codesamples>
<caption>Sample Code Sample
<xmp>
<litdata object=codesample1>
</xmp>
</figure>
```

SGML Definition

```
    CLASS           NAME        #IMPLIED
                                -- Can be required on some elements --
```

ClassDef (element class definition)

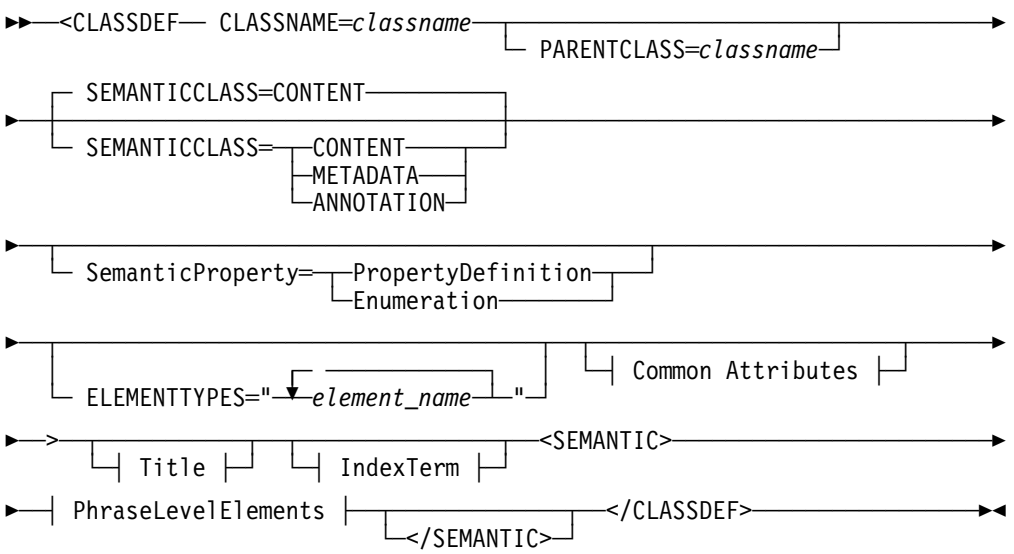
Purpose

ClassDef defines an element class.

Valid Contexts

PropertyDefinitions.

Syntax



Attributes and Contained Elements

CLASSNAME=*classname*

The name of the class being defined. Class names have the same scoping rules as element IDs, but are separate from element IDs.

PARENTCLASS=*classname*

The name of a class that is the parent of this class. Using ParentClass=, you can define class hierarchies that represent the logical structure of the things your information is about.

SEMANTICCLASS=CONTENT | METADATA | ANNOTATION

Defines the base semantic class for the element. In the InfoMaster architecture, all information is one of three classes, either content, metadata or annotation. In most cases, the semantic class for an element is determined by what contains it and does not need to be specified. However, there may be instances where you want to associate a specific semantic class with an element class. The typical use of SemanticClass= is to assign a semantic class of annotation. For example, you can make a paragraph into annotation by creating a new class and specifying a semantic class of annotation.

ElementTypes=*element names*

Defines those element types (generic identifiers) to which this ClassDef applies. Use ElementTypes= when a ClassDef is only meaningful for a specific set of element types.

Title

An optional descriptive title. The title may be used, for example, in selection lists presented by authoring systems for selecting element classes.

IndexTerm

An index term associated with this class. When an IndexTerm is defined for a class, the processing system can automatically generate index entries for an element of that class. The class hierarchy defined by the ParentClass= attribute can be used to generate multiple levels of index entries.

Semantic
Defines the semantic meaning of a given class. Semantic is intended to document what a given element class means to the author that defined it.

Usage

Use ClassDef to define element classes that are specific to your information. The most common use of ClassDef is to define new phrase classes. For example, in the documentation of a graphical user interface, you may want to define phrase classes for all the different types of user interface elements in order to make your markup more precise and to enable the automatic generation of indexes for print presentation.

You can associate presentation styles and other processing with specific element classes. Do not define classes that are purely presentational, such as Bold or Italic. Define classes that have a specific meaning.

Examples

```
<prolog>
  ⋮
<PropertyDefinitions>
 <PropertyDef version='v2r3'>
 <ClassDef ElementType=figure classname=codesamples>
  <title>Code Samples
  <IndexTerm>code sample
  <semantic>Figures that contain code samples.
 </ClassDef>
</PropertyDefinitions>
  ⋮
</prolog>
```

SGML Definition

```
<!ENTITY % ClassDef         "Title?, IndexTerm?, Semantic" >

<!ELEMENT  ClassDef         - - (%ClassDef;)      -- Class Definition -->
<!ATTLIST  ClassDef
    ClassName        NAME         #REQUIRED
    ParentClass      NAME         #IMPLIED    -- Parent of this class --
    ElementTypes     NAMES        #IMPLIED -- GIs this Def applies to --
    SemanticClass    NAME        -- (Content | MetaData | Annotation) --
                                  MetaData
                                  -- The semantic class of this class --
    SemanticProperty (PropertyDefinition | Enumeration)
                                  #IMPLIED
    %Common.Atts;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Content"
    InfoMaster       NAME         #FIXED    "PropertyDefiniton"
>
<!ELEMENT  Semantic         - O (%DataPool;)* >
<!ATTLIST  Semantic
    %Common.Atts;

    InfoMasterSemanticClass
```

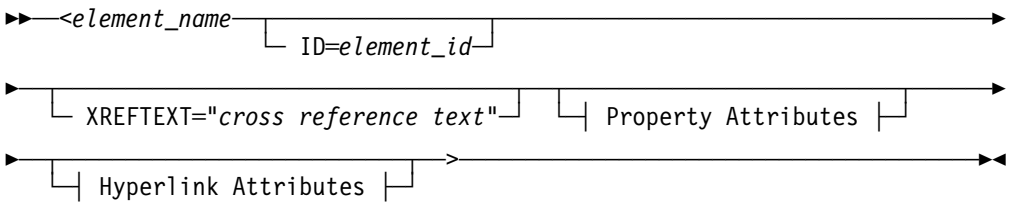
```
                        NAME          #FIXED    "Annotation"
>
```

Common Attributes

Purpose

The common attributes are valid for almost all elements. The common attributes include the ID= attribute, various property definitions, and several hyperlink attributes.

Syntax



Usage

ID=*element_id*

Defines a unique identifier for an element. Any element can have an ID, and some elements must have an ID. Elements for which the ID= attribute is required show the ID= attribute in their syntax definitions.

Any element that has an ID can be the target of any kind of hyperlink.

XREFTEXT=*cross reference text*

Defines the text to be used when the element is the target of a hyperlink that generates a reference, such as XRef. Use XRefText= on elements that do not normally have a title or that contain more than just a phrase. For example, if you want to make a paragraph a link target, you could use XRefText= to define a meaningful reference to the paragraph. When an untitled element is refered to and there is no cross reference text defined, only the location indicator can be generated.

Migration Note

The common XRefText= attribute replaces the SPOT element, as any element can now take an ID and thus be a reference target.

Property Attributes

These attributes define various properties of an element. See “Property Attributes (element properties)” on page 268 for details.

HyperLink Attributes

These attributes define various types of hyperlinks from one element to another. See “Hyperlink Attributes (hyperlink definitions)” on page 140 for details.

SGML Definition

```
<!--Common.Atts.Base has all common attributes except those which
    are specified by any element in the DTD.  These attributes are:
        HyNames
        LexModel
        RefType
        ContentRef
        ID
-->

<!--The following entities are used where these attributes are overriden
    by specific elements to include the common specifications for the
    attribute.
-->

<!ENTITY   Common.HyNames
    "linkends LinkEnds anchrole AnchorRoles conloc ContentRef"
>

<!ENTITY   Common.LexModel
    "Version         VersionSpec
     Level           VersionSpec
     HWPlatform      VersionSpec
     SWPlatform      VersionSpec
     OutputDevice    VersionSpec
     docroles names"
>

<!ENTITY   Common.RefType
    "Qualification Qualification"
>

<!ENTITY % Common.Atts.Base
   "XRefText         CDATA        #IMPLIED
    InfoType         CDATA        #IMPLIED
    %Hyperlink.Attributes;
    %Property.Attributes;"
>

<!--Each of the following entities consists of Base, all of the
    excluded attributes but the one to be excluded, indicated in the
    name of the attribute.
-->

<!--The Class attribute defined here is used for local additions to
    this DTD.  It is our intent to perform this function using
    SUBDOC to isolate these additions and using architectural
    forms to define the processing of the element when this is
    technically possible (these functions are supported by vendors).
    concepts).
-->

<!ENTITY % Common.Atts.NoConRef
   "ID               NAME         #IMPLIED
    HyNames          CDATA        #FIXED    ''  -- no conref att here --
    HyTime           (ilink | hybrid)
                                  hybrid
```

```
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoID
   "ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoLexMdl
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoLexMdl.NoHyTime
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoRefType
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoRefType.NoClass
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    %Common.Atts.Base;"
>
```

Common Attributes

```
<!ENTITY % Common.Atts.NoClass
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    %Common.Atts.Base;"
>
<!ENTITY % Common.Atts.NoLexMdl.NoRefType
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoHyTime
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!-- These attributes are common to BibRecord and LibRecord and
     their derivitives.  They are broken out to add the DescribedDocument
     AnchRole.  They are placed here to facilitate their use in all of
     these elements.
-->
<!ENTITY % LibRecord.Atts
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    XRefText         CDATA        #IMPLIED
    InfoType         CDATA        #IMPLIED
    ContentAssociations
                     NAMES        #IMPLIED
```

```
    AnchorRoles      NAMES        #IMPLIED
    LinkEnds         NAMES        #IMPLIED
    %Hyperlink.Attributes.BookMaster;
    %Property.Attributes;"
>
```

Valid Contexts

PropertyDef.

ControlInfo (process-related control information)

Purpose

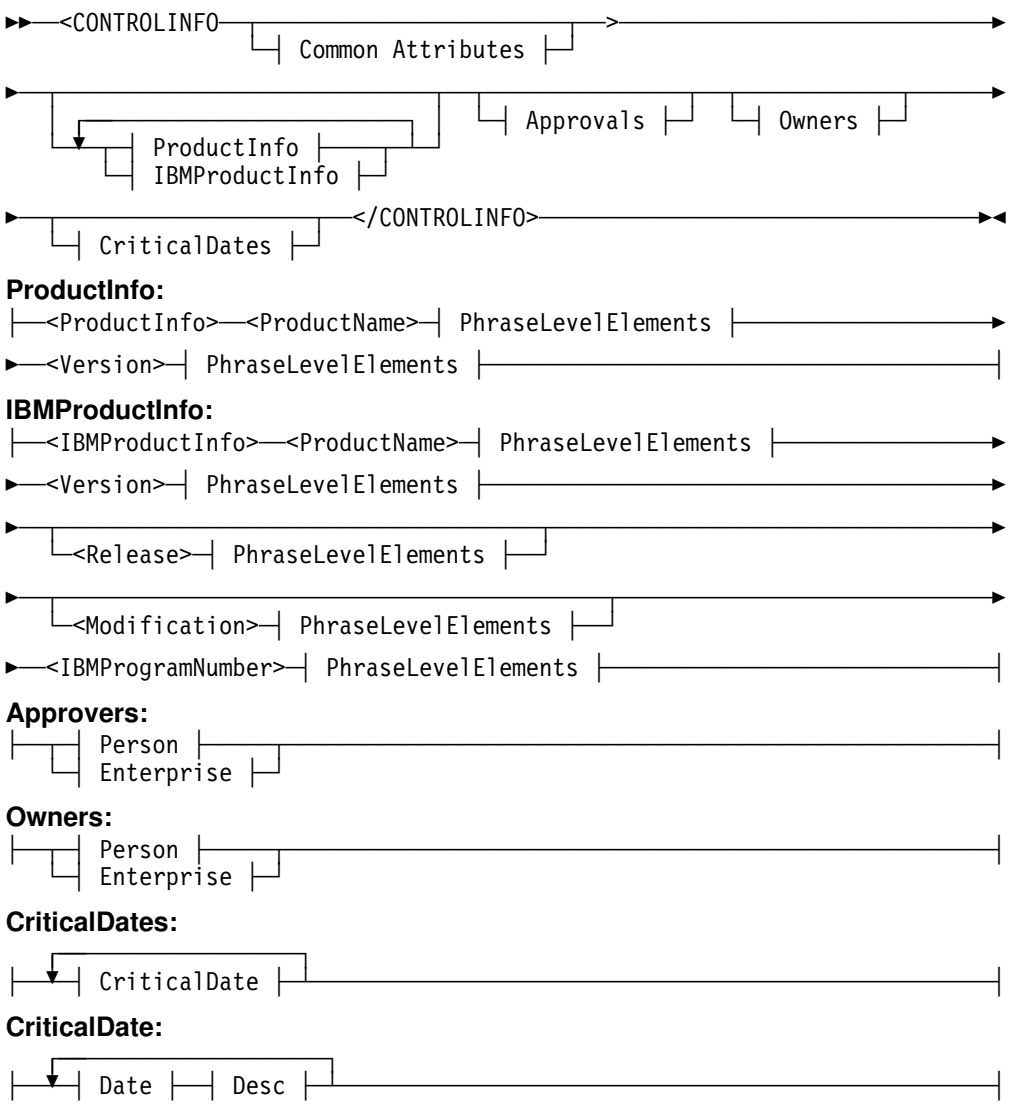
Contains information need for the purpose of applying process and revision control to documents, such as version and release information, critical dates, lists of approvers and owners, and so on. Much of this information is intended to support the needs of ISO 9000-compliant processes.

Valid Contexts

DocTrackingInfo.

Syntax

ControlInfo (process-related control information)



Attributes and Contained Elements

ProductInfo

Contains the information about a non-IBM product associated with the document. ProductInfo contains the following elements:

ProductName

The name of the product.

Version

The product version number

IBMProductInfo

Contains the information about an IBM product associated with the document. IBMProductInfo contains the following elements:

ProductName

The name of the product.

Version
The product version number

Release
The product release number

Modification
The product modification level

IBMProgramNumber
The program number.

You can have as many ProductInfo elements as necessary.

Approvers
Contains a set of Person or Enterprise elements defining who must approve the document.

Owners
Defines the owner or owners of the document.

CriticalDates
Describes any critical dates related to the document, such as draft dates, release dates, approval dates, removal dates, and so on.

Usage

Use ControlInfo to contain information about a document used to control its distribution and use.

Examples

```
<Prolog>
<DocTrackingInfo>
  ⋮
<ControlInfo>
<IBMProductInfo>
<ProductName>IBM Whizbang-O-Matic
<version>1<release>1<modification>0</version>
</IBMPproductInfo>
<Approvers>
<person>
<name><last>Cheese<first>Big
</person>
</Approvals>
<Owners>
<name><last>Cheese<first>Bigger
</Owners>
</CriticalDates>
<CriticalDate
<date>12 June 94
<desc>Approval</desc>
</criticaldate>
<CriticalDate>
<date>11 June 94
<desc>Distribute approval draft
</criticaldate>
</CriticalDates>
</ControlInfo>
  .
```

```
  :
</Prolog>
```

SGML Definition

```
<!ENTITY % ControlInfo
    "(%ProductInfo;)*,
     Approvers?, Owners?,
     CriticalDates?"
>

<!ELEMENT  ControlInfo      - - (%ControlInfo;)
                                    -- Document Control Information --
>
<!ATTLIST  ControlInfo
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  RetrievalKey     - O (%DataPool;)* >
<!ATTLIST  RetrievalKey
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  ProductInfo      - O (ProductName, Version?) >
<!ATTLIST  ProductInfo
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT (ProductName | Version)
                           - O (%DataPool;)*
>
<!ATTLIST (ProductName | Version)
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  IBMProductInfo   - O (ProductName,
                                 Version?, Release?, Modification?,
                                 IBMProgramNumber?)
>
<!ATTLIST  IBMProductInfo
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT (Release | Modification | IBMProgramNumber)
                           - O (%DataPool;)*
>
<!ATTLIST (Release | Modification | IBMProgramNumber)
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  Approvers        - O (Enterprise | Person)+ >
<!ATTLIST  Approvers
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  Owners           - O (Enterprise | Person)+ >
<!ATTLIST  Owners
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  CriticalDates    - O (CriticalDate)+ >
<!ATTLIST  CriticalDates
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  CriticalDate     O O (Date, Desc) >
<!ATTLIST  CriticalDate
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

CopyRight (copyright attribution)

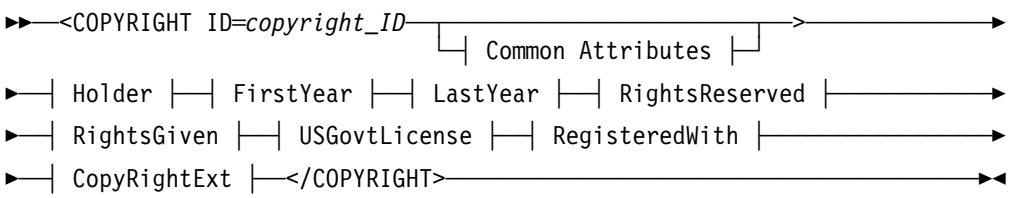
Purpose

Defines the copyright information for a single copyright holder.

Valid Contexts

CopyRights.

Syntax



Attributes and Contained Elements

ID=*copyright_id*
The ID of the CopyRight element. ID is required for CopyRight.

Holder
The holder of the copyright.

FirstDate
The first date that the copyright was claimed.

LastDate
The last date that the copyright was claimed.

RightsReserved
Any rights explicitly reserved. This text may be generated rather than held explicitly.

RightsGiven
Any rights given, for example, the right to copy certain parts as long as the proper copyright attribution is used.

USGovtLicense
Indicates that the standard license for use by the United States government applies. Normally this element is empty with the text generated.

CopyrightExt
Any extension to the copyright not covered by RightsReserved or RightsGiven.

RegisteredWith
Names the agencies with which the copyright has been registered.

Usage

Specify one CopyRight for every copyright holder for the document or division. You must specify at least the copyright holder and the first date. The presentation of the copyright statement in the final document is a function of the output style.

Examples

```
<prolog>
<CopyRights>
<CopyRight id=ibmprimary>
<Holder>IBM Corporation
<FirstYear>1992
<LastYear>1992
<RightsGiven>
<RightsReserved>All Rights Reserved
<USGovtLicense>
<RegisteredWith>U.S. Government Copyright Office
<CopyRightExt>
</CopyRight>
</CopyRights>
```

SGML Definition

```
<!ELEMENT  CopyRight        - -
                      (Holder, FirstYear, LastYear?, RightsReserved?,
                       RightsGiven?, USGovtLicense?,
                       RegisteredWith?, CopyRightExt?)
>
<!ATTLIST  CopyRight
    %Common.Atts;
>

<!ELEMENT  Holder           - O (%DataPool;)* >
<!ATTLIST  Holder
    %Common.Atts;
>

<!ELEMENT  (FirstYear | LastYear)
                            - O (%DataPool;)*
>
<!ATTLIST  (FirstYear | LastYear)
    %Common.Atts;
>

<!ELEMENT  RightsReserved   - O (%DataPool;)* >
<!ATTLIST  RightsReserved
    %Common.Atts;
>

<!ELEMENT  RightsGiven      - O (%DataPool;)* >
<!ATTLIST  RightsGiven
    %Common.Atts;
>

<!ELEMENT  USGovtLicense    - O (%DataPool;)* >
<!ATTLIST  USGovtLicense
    Auto             (Auto)       #CONREF
    %Common.Atts;
>

<!ELEMENT  RegisteredWith   - O (%DataPool;)* >
<!ATTLIST  RegisteredWith
    %Common.Atts;
>

<!ELEMENT  CopyrightExt     - O (%InfoUnitPool;)*
                                              -- Copyright extension --
>
<!ATTLIST  CopyrightExt
    %Common.Atts;
>
```

CopyRights (copyright attributions)

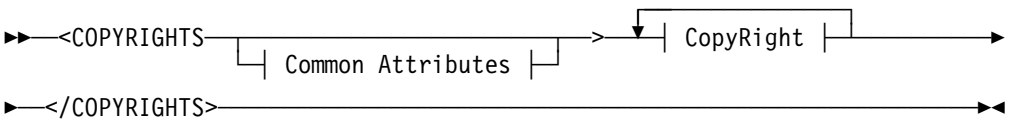
Purpose

Contains copyright attributions for a document or division. Copyright attributions define the intellectual property rights held in the information contained by the document.

Valid Contexts

Prolog, DivProlog.

Syntax



Attributes and Contained Elements

CopyRight

The CopyRight element contains a single copyright attribution for the document or division.

Usage

Use CopyRights to contain the copyright ownership information for the document or division. Put copyrights at the highest level to which they apply. For example, the primary author of the document should always have a copyright attribution at the document level, but a portion of a document may be copyrighted by someone else. For example, Unix™ documentation often has parts copyrighted by other entities than the document author.

Examples

```
<prolog>
<CopyRights>
<CopyRight id=ibmprimary>
<Holder>IBM Corporation
<FirstDate>1992
<LastDate>1992
<RightsGiven>
<RightsReserved>All Rights Reserved
<USGovtLicense>
<RegisteredWith>U.S. Government Copyright Office
<CopyRightExt>
</CopyRight>
</CopyRights>
```

SGML Definition

```
<!ELEMENT  CopyRights       - - (CopyRight+) >
<!ATTLIST  CopyRights
    %Common.Atts;
    InfoMethod       NAME         #FIXED    "ContainingDivision"
>
```

Danger (danger notice)

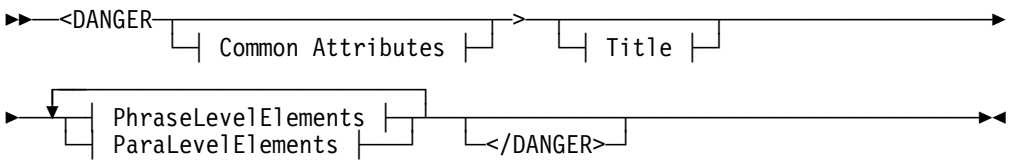
Purpose

Contains a danger notice.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

Title
In the default style, the generated English language title for Danger is 'Danger:'. When specified, the Title replaces any generated title.

PhraseLevelElements
ParaLevelElements
Danger has the same content model as paragraphs, except that Danger can contain paragraphs

Usage

Use Danger to create a danger notice, consisting of one or more paragraphs or other paragraph-level elements. Dangers are normally used to warn about actions that may cause injury or death to a person.

Examples

Simple example:

```
<danger>Working under an automobile supported only by the jack
may result in injury or death.
Always use jack stands or ramps in axle pairs to support your
vehicle.
</danger>
```

Example with title:

```
<Danger><title>Danger title</title>
Danger content
```

SGML Definition

```
<!ELEMENT  Danger           - O (Title?, NoteBody) >
<!ATTLIST  Danger
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>
```

Date

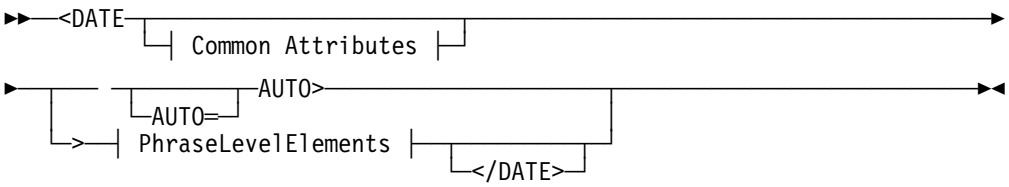
Purpose

The Date element contains a date, either directly, or as a content reference using the Auto attribute. When Auto is used, the date that is presented is defined by the active document style.

Valid Contexts

PhraseLevelElements, Approvals, CriticalDates.

Syntax



Attributes and Contained Elements

AUTO

Indicates that the presented date is to be defined by the presentation system as a result of the active document style. By default, this is the system date at the time the document is processed.

Usage

Use the Date element to contain a date. In addition to being valid where ever PhraseLevelElements is valid, Date is required in some of the ControlInfo elements, such as Approvals and CriticalDates.

IBM ID Doc does not define the format of the date element, but processing systems may define format constraints for Date element content. The Style= attribute can be used to define style names for specific date formats that are understood or enforced by specific processing systems.

Examples

```
<annot>
<p>This change was made on <date>14 Dec 1992</date>.
</annot>
```

SGML Definition

```
<!ELEMENT  Date             - O (#PCDATA) >
<!ATTLIST  Date
    Auto             (Auto)       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
>
```

Decimal (decimal number)

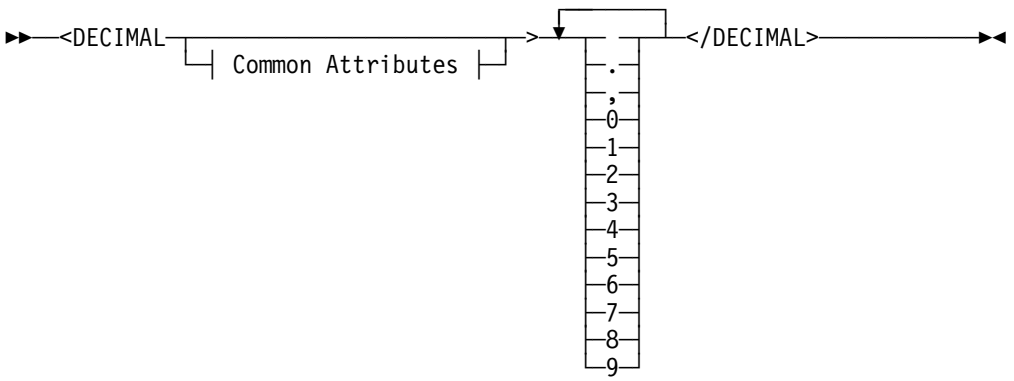
Purpose

The Decimal element identifies decimal data.

Valid Contexts

PhraseLevelElements.

Syntax



Usage

Use the Decimal element to identify decimal data. Decimal data is data that is encoded in a base 10 numbering system. Its content is the ten decimal digits, '0' thru '9', plus periods (.), commas (,), and spaces. A period may be used to express decimal fractions, e.g. 10.5. A space character in the content of this number represents a visual separator within a single decimal number (not a separator between multiple numbers). Spaces may be used to separate units from thousands, e.g. 24 000. A comma may also be used for this purpose: e.g. 24,000.

Examples

```
<bin>11000001</bin> = <decimal>193<decimal>
```

SGML Definition

```
<!ELEMENT  Decimal          - - (#PCDATA)      -- decimal data phrase -->
<!ATTLIST  Decimal
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "DecimalData"
    LexModel         CDATA        #FIXED
                          "#CONTENT (digits | '.' | ',' | ' ')*
                           &Common.LexModel;"
>
```

DecisionPoint

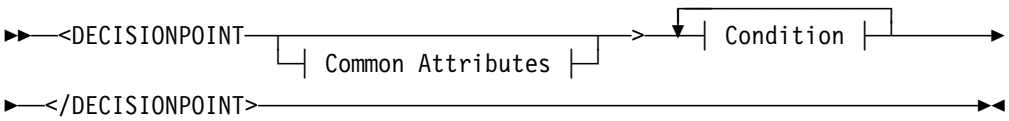
Purpose

Defines one or more condition/action pairs defining what the next step should be at any point within a procedure.

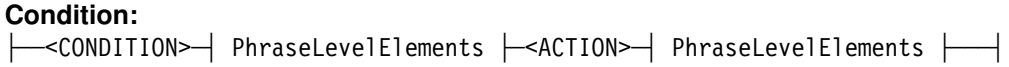
Valid Contexts

ProcStep, ProcExit.

Syntax



Condition:



Attributes and Contained Elements

CONDITION

Defines a condition, for example “step didn't work”.

ACTION

Indicates what action to take for the condition.

Usage

Use decision point to precisely define actions to take under specific conditions.

Examples

```
<Procedure>
  ⋮
 <ProcStep id=procstepB>
   <title>Second thing you do
   <ReferenceKeys>rk1 rk6 <refkey>C</>
   <Desc><p>Do even more stuff.
   <DecisionPoint>
     <Condition>You are done
     <Action>Go to <XRef refid=procexita>.
     <Condition>You are not done
     <Action>Go to <XRef refid=procstepA>.
   </DecisionPoint>
  ⋮
</Procedure>
```

SGML Definition

```
<!ELEMENT  DecisionPoint    - - (Condition, Action)+ >
<!ATTLIST  DecisionPoint
    %Common.Atts;
>

<!ELEMENT  Condition        - O (%DataPool;)* >

<!ELEMENT  Action           - O (%DataPool;)* >

<!ATTLIST  (Condition | Action)
    %Common.Atts;
>
```

DefinedTerm

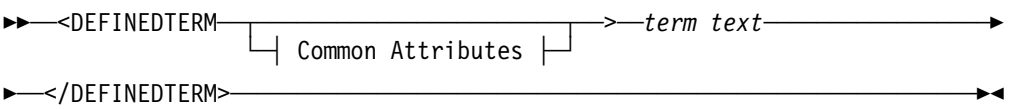
Purpose

DefinedTerm contains a term and defines its hyphenation points, normally within a glossary entry. When used outside the context of GLEntry, DefinedTerm identifies a term that has been defined elsewhere. is valid.

Valid Contexts

DLEntry, GLEntry, PhraseLevelElements.

Syntax



Attributes and Contained Elements

term text
The text of the term. Within the DefinedTerm element, hyphens (‐) define the allowed hyphenation points. To indicate required hyphens, use the &reqhyph; entity. In the default style, the hyphens used to define hyphentation are not presented.

Usage

Use DefinedTerm to define a word as a term and indicate its hyphenation points. Within the context of GLEntry, the DefinedTerm start and end tags can be omitted.

Examples

```
<glentry id=apple>
<DefinedTerm>ap-ple
<gd>A fruit of the apple tree.
</glentry>
  ⋮
<p>An <DefinedTerm contentref=apple> a day keeps the doctor away.
```

SGML Definition

```
<!ELEMENT  DefinedTerm      O O (%DataPool;)* >
<!ATTLIST  DefinedTerm
    SearchClass      NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinedTerm"
    Notation         (Term)
                                  #FIXED    "Term"
>
```

Delim (syntactic delimiter)

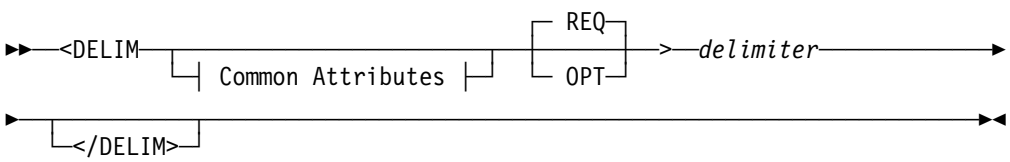
Purpose

Defines a delimiter within a syntax definition group or fragment. Delimiters normally occur in pairs and delimit subexpressions within a main expression.

Valid Contexts

Group, RepSep, SyntaxPhrase.

Syntax



Attributes and Contained Elements

REQ | OPT
Indicates whether or not the delimiter is optional. REQ (required) is the default. Any delimiter that is not optional is by definition required.

delimiter
The delimiter characters.

Usage

Use Delim to define delimiters within a syntax definition. Delimiters mark the start and end of subexpressions within a main expression and usually occur in pairs. Typical separators are quotation marks and parenthesis ('()'). Delimiters are distinquished from separators in that separators simply separate one thing from another, rather than enclosing something.

Examples

```
<syntax>Attribute Specification
  <group>Attspec
    <sep>&ssbl;
    <kwd>ATTNAME
    <oper>=
    <delim>"
    <var>value
    <delim>"
  </group>
</syntax>
```

SGML Definition

```
<!ELEMENT  Delim            - O (%DataPool;)*     -- Syntax Delimiter -->
<!ATTLIST  Delim
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;
>
```

Desc (element's description)

Purpose

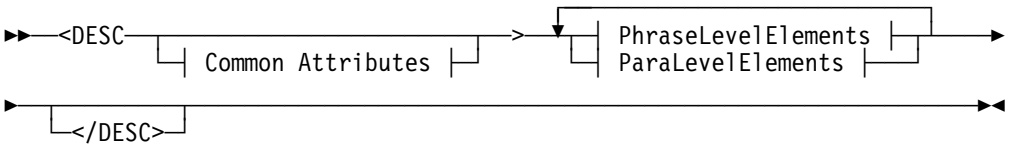
Desc contains a description of an element. Many elements allow Desc in their content, usually as the first element (or following the title) or the last element.

Valid Contexts

Many elements.

DescribedDocument Anchor Role (link to a described document)

Syntax



Attributes and Contained Elements

PhraseLevelElements
ParaLevelElements
Desc can contain anything except divisions.

Usage

Desc is intended to contain a description of the content of the element within which Desc appears. The presentation effect of a given Desc element is determined by the style of the element that contains the Desc. For example, within a figure, the Desc may be presented as part of the figure caption, but in another element the Desc may not be presented at all in the default style.

Examples

```
<figure id=figa>Title of Figure
<caption>This is the figure caption
<desc>This is the figure description.
<figseg>
</figseg>
</figure>
```

SGML Definition

```
<!ELEMENT  Desc             - O (%InfoPool;)*      -- Description -->
<!ATTLIST  Desc
    %Common.Atts;
    InfoMaster       NAME         #FIXED    "Description"
>
```

DescribedDocument Anchor Role (link to a described document)

Purpose

When specified as a value of the AnchorRoles= attribute, indicates that the cooresponding Linkends= ID value refers to the actual document being described. The DescribedDocument role is intended only for use from the bibliography record elements, but could be used with other elements.

Usage

Use the DescribedDocument role in conjunction with Linkends= to refer to the entity that represents the document being described. The entity can represent any representation of the document, not necessarily the SGML source of the document. It is up to the application or presentation system resolving the link to determine how to process or present a document linked to in this way. It is expected that for typical online presentation systems, the result will be to open the target document, if possible. This link would normally be ignored for print processes.

Examples

In the following example, a BibRecord element uses the DescribedDocument role to link to the online version of the document:

```
<!DOCTYPE IBMIDDOC PUBLIC "+//ISBN 0-933186::IBM//DTD IBMIDDoc//EN"
 [
  <!ENTITY  SGMLHandbook "+//ISBN 0-19::Goldfarb//NONSGML
                          ISBN 0-19-853737-9//EN//Online"
                          NDATA BookManager>
 ]>
<IBMIDDoc>
 <prolog>
  <bibrecords>
   <bibrecord   linkends="SGMLHandbook" HyTime=ilink
              AnchorRoles="LinkSource DescribedDocument">
     ...
   </bibrecord>
  </bibrecords>
 </prolog>
  ⋮
</ibmiddoc>
```

Note that the first role name, linksource, refers to the BibRecord element itself, indicating that it is the source of the link.

Div (hierarchical division)

Purpose

Division defines the hierarchical organization of the information.

Migration Note

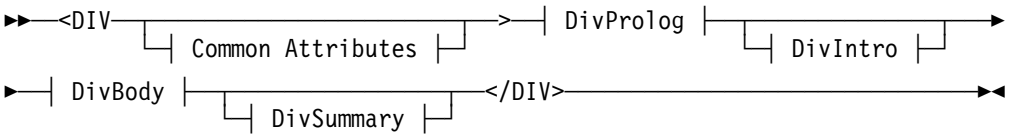
DIV replaces all the Hx elements from DCF BookMaster.

Valid Contexts

Body, Part, DivBody, DivIntro, DivSummary, WebNodeBody.

Div (hierarchical division)

Syntax



Attributes and Contained Elements

DivProlog
Contains the division title and any other descriptive information, such as property definition elements, tracking information, and so on.

DivIntro
Contains an optional division introduction. Divintro is normally untitled, but can itself contain divisions.

DivBody
Contains the actual division content. Normally, the DivBody tag is implied automatically.

DivSummary
Contains an optional division summary. DivSummary is normally untitled, but can itself contain divisions.

Usage

Hierarchical divisions contain, normally, a title followed by contained elements. In the formal definition of IBM ID Doc, the division contains two required subparts, the Division Prolog and the Division Body, and can contain optional introduction and summary elements. However, because of the way IBM ID Doc is defined, you normally never have to actually enter the DivProlog or DivBody tags, as they are implied automatically. The division title, what you may normally think of as the "heading," is the first element of DivProlog and is required. Following the title, the DivProlog can contain Prolog elements such as TrackingInfo, LinkDescriptions, PropertyDefinitions, GlossaryTerms, and TrademarkDefinitions.

You must explicitly end the Div element in order to start another Div at the same hierarchical level. This is because the hierarchical level of each division is defined by the containment structure, not by explicit tag names.

Examples

Simplest markup, no optional DivProlog elements:

```
<div>About Hierarchical Divisions
<p>Hierarchical divisions define the logical organization of
a document.
```

In the above example, the DivProlog, Title, and DivBody tags are omitted because they are implied automatically. The fully-expanded markup for the example is:

```
<div><divprolog><title>About Hierarchical Divisions</title></divprolog>
<divbody>
<p>Hierarchical divisions define the logical organization of
a document.
```

Div must be explicitly ended in order to start a new Div element at the same level:

```
<div>About Hierarchical Divisions
<p>Hierarchical divisions define the logical organization of
a document.
<!-- -->
<div>Introducing Hierarchical Divisions
<p>This is the introduction.
</div>
<!-- -->
<div>Explaining Hierarchical Divisions
<p>This is the explanation.
</div>
</div>
```

The two Div elements “Introducing...” and “Explaining...” are contained within “About Hierarchical Divisions” and they are both at the same hierarchical level, like so:

About Hierarchical Divisions
    Introducing Hierarchical Divisions
    Explaining Hierarchical Divisions

You can include local property definitions and other DivProlog elements like so:

```
<div>About Hierarchical Divisions
<PropertyDefinitions>
<dldef search=divisions>
</PropertyDefinitions>
<p>Hierarchical divisions define the logical organization of
a document.
```

SGML Definition

```
<!ENTITY % Division         "DivProlog, DivIntro?, DivBody, DivSummary?">

<!ENTITY % SpecialDiv
    "SpecialDivProlog, DivIntro?, DivBody, DivSummary?"
>

<!ELEMENT  Div              - - (%Division;) -- Hierarchical Division -->
<!ATTLIST  Div
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

DivBody (division body)

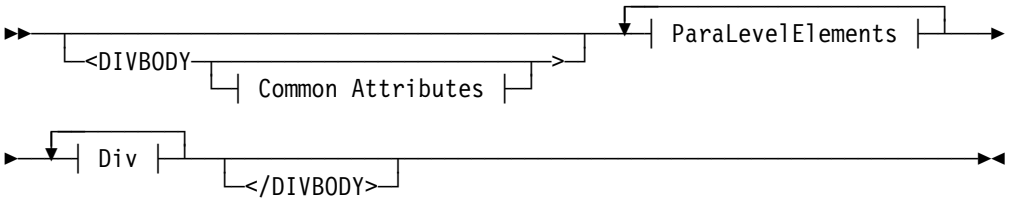
Purpose

DivBody contains the main content of a hierarchical division.

Valid Contexts

Div.

Syntax



Attributes and Contained Elements

ParaLevelElements and Div

Contained elements, both divisions and paragraph-like elements.

Usage

DivBody contains the content of a hierarchical division. It is only necessary to explicitly specify the DivBody start tag when the hierarchical division is completely empty. Under normal circumstances, the DivBody start tag is implied automatically.

Examples

See “Div (hierarchical division)” on page 97 for examples.

SGML Definition

```
<!-- Permissive model required to allow Div and Procedure to mingle. -->
<!ELEMENT  DivBody          O O (%InfoUnitPool; | Div)*
                                                    -- Division Body --
>
<!ATTLIST  DivBody
    %Common.Atts;
    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

DivIntro (division introduction)

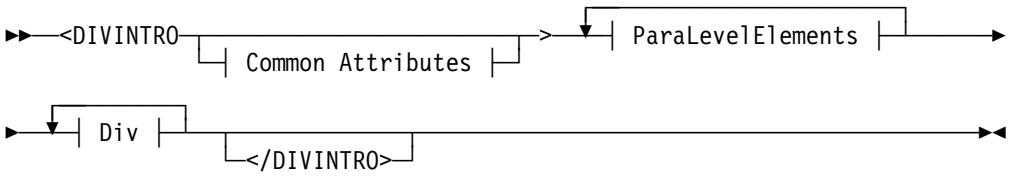
Purpose

Contains the introduction to a hierarchical division (Div).

Valid Contexts

Div, following DivProlog.

Syntax



Attributes and Contained Elements

ParaLevelElements and Div

Contained elements, both divisions and paragraph-like elements.

Usage

DivIntro contains the introduction to a hierarchical division. For example, at the highest level of a document, you may want introductory information to precede the first subdivision.

Migration Note

The use of DivIntro, together with the Bridge element, removes the need to skip heading levels in DCF BookMaster.

In general, DivIntro is useful when you have nested divisions with a Div and you want to introduce the sub-divisions. This sort of structure usually looks something like this:

```
Chapter 1. Getting to Know the Otter

  In this chapter, you will be introduced to the otter,
  one the cutest mammals around.  After reading this
  section, you will be able to distinquish river otters
  from sea otters and American otters from European otters.

  1.1 Recognizing Otters

   ...

  1.2 Varieties of Otters

   ...
```

In this example, the paragraph preceding section 1.1 is an introduction to the entire division and is best put into a division introduction element. The markup for the above example would look like this:

DivIntro (division introduction)

```
<div id=get-to-know-otters>Getting to Know the Otter
 <divintro>
  <p>In this chapter, you will be introduced to the otter,
  one the cutest mammals around.  After reading this
  section, you will be able to distinquish river otters
  from sea otters and American otters from European otters.
 </divintro>
 <divbody>
 <div>Recognizing Otters
  <divbody>
     ...
  </divbody>
 </div>
 <div>Varieties of Otters
  <divbody>
     ...
  </divbody>
 </divbody>
</div>
```

Note how the DivIntro element helps to clearly identify the introductory material as introductory, and makes the division body consist of nothing but nested divisions, rather than mixed paragraphs and nested divisions. This can be useful both in online environments where, for example, you might want to suppress the introductory material to give a more modular presenation, or present only the introductory material to provide a sort of overview of the document contents.

Another feature of division intro is the ability to use divisions within DivIntro to subdivide the introductory material without confusing the division hiearchy. Consider this version of the forgoing example:

```
  Chapter 1. Getting to Know the Otter

    In this chapter, you will be introduced to the otter,
    one the cutest mammals around.  After reading this
    section, you will be able to distinquish river otters
    from sea otters and American otters from European otters.

Where to Go Next

      After you have read this section, see "More about River Otters"
      on page ...

    To Learn More

      To learn more about otters, try these books:

      "Our Friend the Otter" ...
      "More Than You Ever Wanted to Know about Otters" ...

    1.1 Recognizing Otters

     ...

    1.2 Varieties of Otters

     ...
```

The markup for the above would be:

```
<div id=get-to-know-otters>Getting to Know the Otter
 <divintro>
  <p>In this chapter, you will be introduced to the otter,
  one the cutest mammals around.  After reading this
  section, you will be able to distinquish river otters
  from sea otters and American otters from European otters.
  <div>Where To Go Next
    ...
  </div>
  <div>To Learn More
    ...
  </div>
 </divintro>
 <divbody>
```

Note that regular Div elements are used to subdivide the DivIntro section, but that they are not numbered like the divisions in the division body. This is possible because the divisions within the division introduction are structurally distinquished from the divisions in the division body.

Examples

```
<div>First Document Part
<divintro>
<p>The first part talks about two major aspects of the universe,
space and time.
<div>Space
<p>This introduces space.
</div>
<div>Time
This introduces time
</div>
</divintro>
<!-- DivBody element implied by context -->
<div>All About Space
</div>
<div>All About Time
</div>
</div>
```

SGML Definition

```
<!ELEMENT  DivIntro         - O (Title?, TOC?, (%InfoUnitPool;)*, Div*)
                                          -- Division Introduction --
>
<!ATTLIST  DivIntro
    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
    %Common.Atts;
>
```

DivProlog (division metainformation)

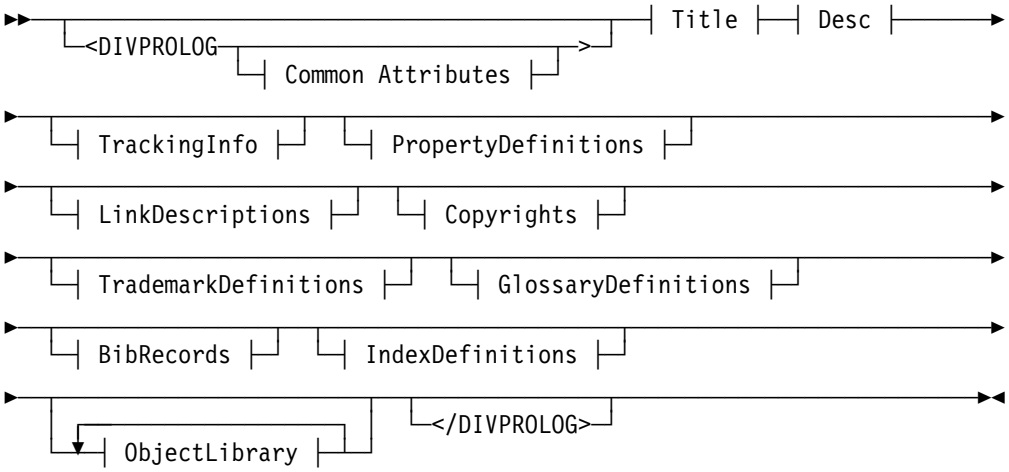
Purpose

DivProlog contains all the descriptive metainformation about a division, including the division title.

Valid Contexts

Div

Syntax



Attributes and Contained Elements

Title
The division title. Normally the Title start tag is implied by the title text itself. When a division has no title the Title start tag must be specified explicitly.

TrackingInfo
Contains the tracking and control information for the division, such as revision definitions, applicable change and control document references, ownership, security classifications, and so on.

PropertyDefinitions
All property definitions applicable to the hierarchical division. Property definitions with no ID= value specified define the global defaults for the division.

LinkDescriptions
Contains LinkDesc elements.

Copyrights
Contains copyright definition elements.

TrademarkDefinitions
Contains trademark definitions for use elsewhere in the document.

GlossaryDefinitions
Contains GLEntry elements for use by reference from anywhere in the document.

BibRecords
Contains BibRecord and LibRecord elements used by this division.

IndexDefinitions
Contains index elements (I1, I2, I3) used by reference within the division.

ObjectLibrary
You can use object libraries to organize re-usable elements. Elements within object libraries will only be processed if they are refered to by ContentRef= attributes.

Usage

Examples

Example of ShortTitle element:

```
<div>
<ShortTitle>Short Title</ShortTitle>
This is a really really long Title
<p>This topic has a long title.
```

Example of TrackingInfo element:

```
<div>Example of Change Tracking Info
<!-- DivProlog is implied by TrackingInfo start tag -->
<TrackingInfo>
<RevisionHistory>
<Revision date="12/19/92"
          author=kimber
          id=rev1>
Created this section.
</revision>
<Revision date="12/20/92"
          author=kimber
          id=rev2>
Fixed some typos.
</revision>
</RevisionHistory>
</TrackingInfo>
<p>This paragraph implicitly ends the DivProlog and starts the
DivBody.
```

SGML Definition

```
<!ELEMENT  DivProlog        O O (%DivProlog;)      -- Division Prolog -->
<!ATTLIST  DivProlog
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Metadata"
>
```

DivSummary (division summary)

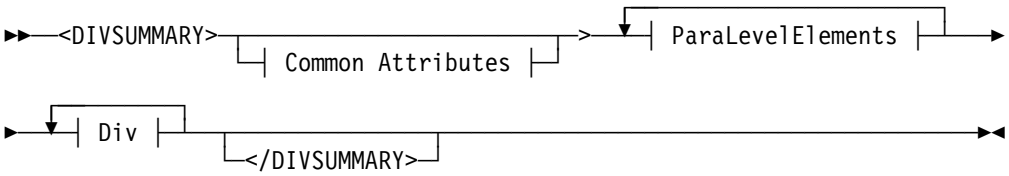
Purpose

Contains the summary of a hierarchical division (Div).

Valid Contexts

Div, following DivBody.

Syntax



Attributes and Contained Elements

ParaLevelElements and Div

Contained elements, both divisions and paragraph-like elements.

Usage

DivSummary contains the summary of a hierarchical division. For example, you can use DivSummary to contain transitions from one division to the next that can be easily suppressed for terse online presentation or given special presentation effects to aid retrieval or navigation.

Examples

```
<div>First of Two Divisions
<p>The content of the first division
<divsummary>
<p>This division discussed x and the next division discusses y.
</divsummary>
</div>
<div>All About Y
</div>
```

SGML Definition

```
<!ELEMENT  DivSummary       - O (Title?, (%InfoUnitPool;)*, Div*)
                                                -- Division Summary --
>
<!ATTLIST  DivSummary
    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
    %Common.Atts;
>
```

DL (definition list)

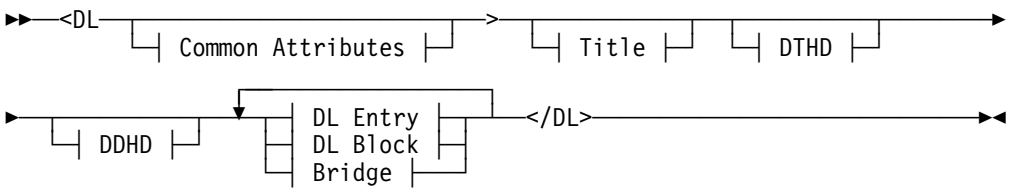
Purpose

Contains pairs of terms and definitions.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

Title
The title of the definition list. When a title is specified, the Title start tag must be specified.

DTHD
The heading for the term part of the definition list. The default presentation style is to use the DTHD as the column heading over the term column.

DDHD
The heading for the definition part of the definition list. The default presentation style is to use the DDHD as the column heading over the definition column.

DLEntry
A single term and its definition. The DLEntry element contains two elements, DefinedTerm and DD (definition description).

DLBlock
Contains a group of DLEntry elements organized for some reason.

Bridge
A Bridge element used to provide a transition or connection between two DLBlocks.

Usage

Use definition lists as a generic definition structure for defining things other than glossary terms. A definition list can be a defining list for a class of objects by naming that class in the CLASS attribute. Entries can be organized within a definition list using definition list block (DLBlock) elements. Bridge elements can also be used to create transitions or connections between blocks of entries. Definition lists have an optional title.

Examples

```
<dl>
<dlentry><DefinedTerm>Term
<dd>Definition
<dlentry>term
<dd>Definition
</dl>
```

SGML Definition

```
<!ELEMENT  DL              - - (Title?, (DTHD, DDHD)?,
                                (DLEntry | DLBlock | Bridge)+)
                                                -- Definition List --
>
<!ATTLIST  DL
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefiningList"
>

<!ELEMENT DTHD             - O (%DataPool;)*
                                     -- Definition List Term Heading --
>
<!ATTLIST DTHD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermHeading"
>

<!ELEMENT DDHD             - O (%DataPool;)*
                                 -- Definition List Definition Heading --
>
<!ATTLIST DDHD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionHeading"
>
```

DLBlock (definition list block)

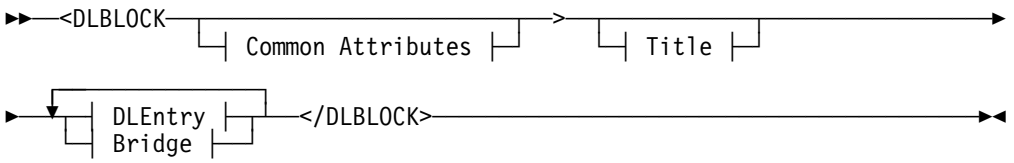
Purpose

Organizes definition list entries within a definition list.

Valid Contexts

DL.

Syntax



Attributes and Contained Elements

Title
The title of the definition block.

DLEntry
Bridge
Any combination of DLEntry and Bridge elements.

Usage

Use DLBlock to organize DL entries arbitrarily. For example, you can use DLBlock to create logical subdivisions within a long definition list, or use two DLBlocks with a bridge between them to highlight some relationship between groups of entries.

Examples

```
<dl class=sensecode><title>Fubar Sense Codes
<dlblock><title>Rabuf Module, Prefix <hex>ABCF</hex>
<dlentry><hex>0000</hex>
<dd>Normal completion
<dlentry><hex>FFFF</hex>
<dd>Really not normal completion
</dlblock>
<dlblock><title>Bafur Module, Prefix <hex>B00F</hex>
  ⋮
</dlblock>
</dl>
```

SGML Definition

```
<!ELEMENT  DLBlock          - O (Title?, (DLEntry | Bridge)+)
                                  -- Definition List Block of Entries --
>
<!ATTLIST  DLBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionBlock"
>
```

DLEntry (definition list entry)

DLEntry (definition list entry)

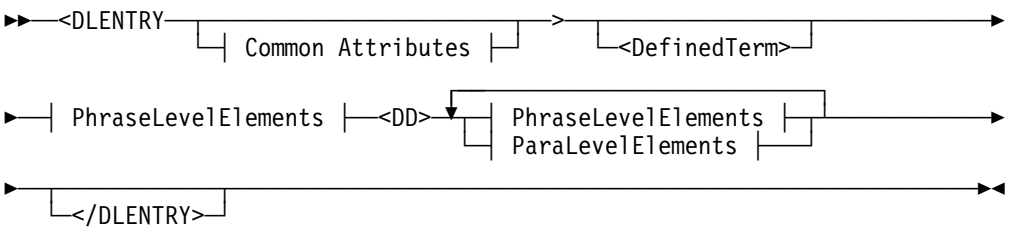
Purpose

Contains a single term and its definition.

Valid Contexts

DL, DLBlock.

Syntax



Attributes and Contained Elements

DefinedTerm

Contains the term being defined by this DLEntry. The DefinedTerm start and end tags can be omitted.

DD

Contains the definition of the term.

Usage

Use the DLEntry element within a definition list (DL) to define information that is not glossary information. Glossary entries (GLEntry) should be used for formal, dictionary-style definitions of words.

Examples

```
<dl>
<dlblock><Title>3-D Concepts
<dlentry>light source angle
<dd>...<dlentry>radiosity
<dd>...
</dlblock>
<dlblock><title>Color Concepts
  ⋮
</dlblock>
</dl>
```

SGML Definition

```
<!ELEMENT  DLEntry          - O (DefinedTerm, DD)
                                        -- Definition List Entry --
>
<!ATTLIST  DLEntry
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Term"
>

<!ELEMENT  DD               - O (%InfoPool;)*
                               -- Definition list term definition --
>
<!ATTLIST  DD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermDefinition"
>
```

DocTitle (document title)

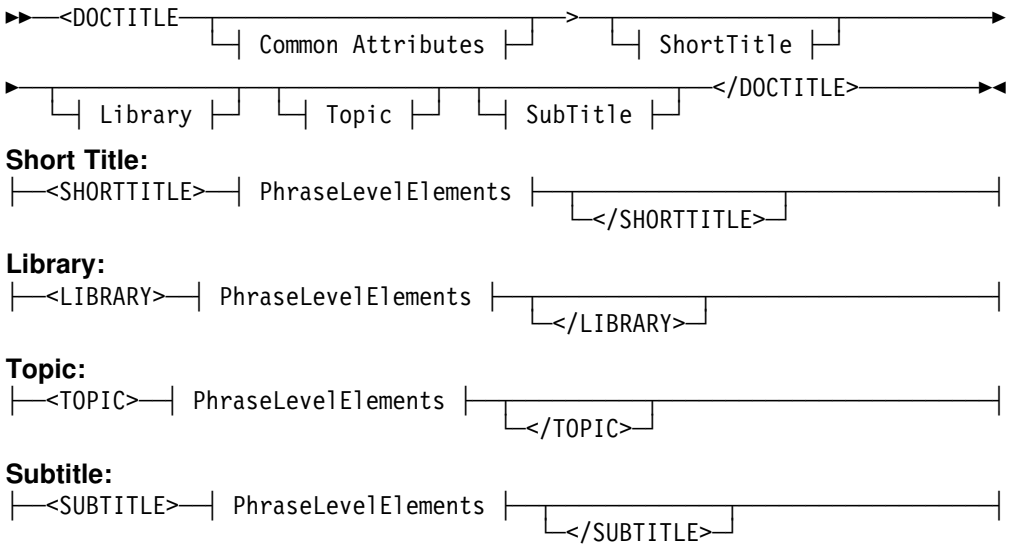
Purpose

Contains and structures the document title information. DocTitle is valid within DocTrackingInfo, BibRecord, and IbmBibRecord elements.

Valid Contexts

BibRecord, IBMBibRecord

Syntax



Attributes and Contained Elements

ShortTitle
Contains the short title for the document. The short title is used when the longer title would be inappropriate, for example, in running feet or window title bars.

Library
The title of the library this document belongs in. The library may also be defined outside the document.

Topic
The topic the book covers, for example “User's Guide”.

Subtitle
A descriptive title of the sort that sometimes appears on title pages, such as “An exploration of the use and misuse of IBM ID Doc”.

Migration Note

The version and release information is now part of ControlInfo.

Usage

The document title organizes the document title. You should always provide a short title for IBM documentation. The short title will be used in places like bookshelf lists, citation references, and the like. The complete title that appears on the document title page page or cover is defined by the document style and may include data from other prolog elements.

Examples

```
<DocTitle><ShortTitle>Bookmaster Reference
<Library>Bookmaster
<Topic>Reference
</DocTitle>
```

SGML Definition

```
<!ELEMENT  DocTitle         - -
                    (ShortTitle?, Library?, Topic, SubTitle?)
                                                -- Document Title --
>
<!ATTLIST  DocTitle
    %Common.Atts;
    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "Title"
    Notation         (LineSpecific)
                                  #FIXED    "LineSpecific"
>

<!-- Note:  ShortTitle is the same element contained by Title -->

<!ELEMENT  (Library | Topic | SubTitle)
                              O O (%DataPool;)*
>
<!ATTLIST  (Library | Topic | SubTitle)
```

```
    %Common.Atts;
>
```

DocTrackingInfo (document tracking and control information)

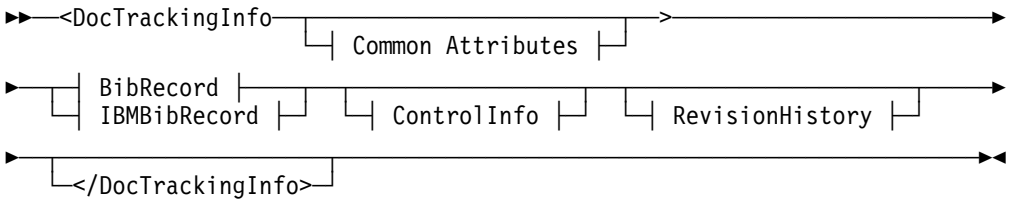
Purpose

Contains all tracking and control information for the document as a whole, including title, author, document control numbers, and so on.

Valid Contexts

Prolog.

Syntax



Attributes and Contained Elements

BibRecord | IBMBibRecord
Contains the document's bibliographic information. Use IBMBibRecord for all IBM product documentation and BibRecord for all other documents.

ControlInfo
Contains all information related to process tracking and control, such as document numbers, program numbers, part numbers, and so on.

RevisionHistory
Contains the revision history information for the document as a whole.

Usage

Use DocTrackingInfo to contain the information needed to completely identify, track, and control a document.

Examples

```
<Prolog>
<DocTrackingInfo>
<BibRecord
<DocTitle>
<ShortTitle>
<Library>
<Topic>
</DocTitle>
<Authors>
  <author id=donday>
   <person>
```

```
   <name><last>Day<first>Don<middle>R<honorific>Mr.</name>
   <address>
   </address>
   </person>
  </author>
  <author id=waynewohler>
   <person>
   <name><last>Wohler<first>Wayne<middle><honorific>Mr.</name>
   <address>
   </address>
   </person>
  </author>
  <author id=drmacro>
   <person>
   <name><last>Kimber<first>W<middle>Eliot<honorific>Mr.</name>
   <address>
   </address>
   </person>
  </author>
 </authors>
</BibRecord>
<ControlInfo>
 <ProductInfo>SGML BookMaster
  <version>1
  <release>0
  <modification>0
  <IBMProgramNumber>5574-234
 </ProductInfo>
 <ibmdocnum>SC32-7891-01
 <isbn>0-89588-806-8
 <ibmpartnum>678543
</ControlInfo>
<RevisionHistory>
 <revision id=release1>
  <desc>First release of the document for review.
 </revision>
</RevisionHistory>
</DocTrackingInfo>
  ⋮
</Prolog>
```

SGML Definition

```
<!ENTITY % DocTrackingInfo
    "(%BibRecord;)?,
     ControlInfo?, RevisionHistory?, QualificationList?"
>

<!ELEMENT  DocTrackingInfo  - O
                    (%DocTrackingInfo;)
                                   -- Document Tracking Information --
>

<!ATTLIST  DocTrackingInfo
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    InfoMasterSemanticClass
```

```
                        NAME         #FIXED    "Metadata"
>
```

EditionNotices

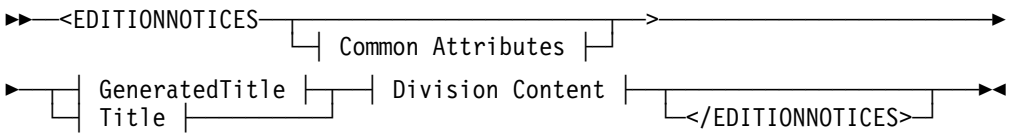
Purpose

EditionNotices is intended to contain the edition notices for the document, including any legally-required statements about intended use, updates, and the like.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the EditionNotices division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

Division Content

EditionNotices has the same content rules as Div elements.

Usage

Examples

```
<frontmatter>
<editionnotices><generatedtitle>
<p>Typical legal googah about editions.
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  EditionNotices   - O (%SpecialDiv;) >
<!ATTLIST  EditionNotices
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

Enterprise (enterprise name and address)

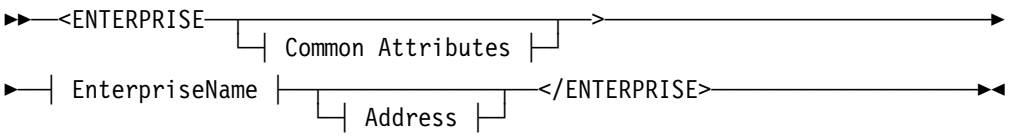
Purpose

Contains the name and address of an enterprise, as opposed to a person.

Valid Contexts

Author, Approvers, Owners.

Syntax



Attributes and Contained Elements

EnterpriseName
Contains the name of an enterprise.

Address
Contains an address.

Usage

The elements Person and Enterprise both contain name and address pairs for use in contexts like Author, Approvers, and Owners where either a person or an enterprise could be meaningful. The main distinction between Person and Enterprise is that Enterprise has a special name element, EnterpriseName that has different subelements than Name does.

Examples

```
<author>
 <enterprise id=IBM-NS>
  <enterprisename>IBM Network Systems Division</enterprisename>
  <address>
  ⋮
  </address>
 </enterprise>
</author>
```

SGML Definition

```
<!ELEMENT  Enterprise       - O (EnterpriseName, Address?) >
<!ATTLIST  Enterprise
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "NameAndAddress"
>
```

EnterpriseName (enterprise name)

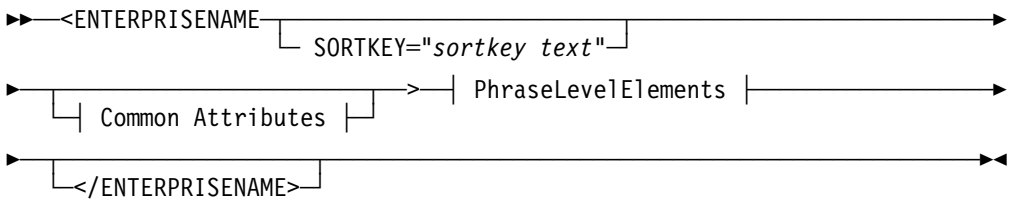
Purpose

Contains an enterprise's name.

Valid Contexts

Enterprise.

Syntax



Attributes and Contained Elements

SORTKEY="*sortkey text***"**
Defines the text string that should be used when sorting the name.

PhraseLevelElements
The enterprise name has no special structuring elements.

Usage

The EnterpriseName simply contains the otherwise unstructured name of an enterprise, such as a company, government agency, or non-profit organization. You can use the SortKey= attribute to control how the enterprise name is sorted.

Examples

```
<author>
 <enterprise id=IBM-NS>
  <enterprisename>IBM Network Systems Division
  </enterprisename>
  <address>
  ⋮
  </address>
 </enterprise>
</author>
```

SGML Definition

```
<!ELEMENT  EnterpriseName   O O (%DataPool;)* >
<!ATTLIST  EnterpriseName
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Name"
>
```

FigSeg (figure segment)

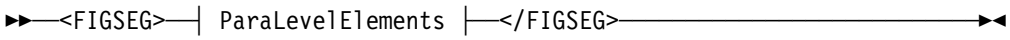
Purpose

The FigSeg element organizes the content of a figure into logical segments. The primary intent of FigSeg is to contain parts of a figure that must be kept together when the figure is presented.

Valid Contexts

Figure.

Syntax



Attributes and Contained Elements

ParaLevelElements

You can place almost anything within a figure segment except divisions.

Usage

Use multiple figure segments to break long exhibit elements into smaller chunks to enable breaking of figures at logical points. In a code sample, for example, you might use one figure segment for each subroutine, ensuring that no subroutines will be broken in the middle.

Examples

Example showing the use of FigSeg to break up a long example.

```
<figure id=figure.b class=codesample>
<caption>A Long Code Sample
<desc>This code sample is very very long.  Each segment
contains a example which contains a reference to an NDATA
entity containing the live code.
<figseg>
<xmp>
<litdata object=codesamp1>
</xmp>
</figseg>
<figseg>
<xmp>
<litdata object=codesamp2>
</xmp>
</figseg>
  ⋮
<figseg>
<xmp>
<litdata object=codesamp10>
</xmp>
</figseg>
</figure>
```

SGML Definition

```
<!ELEMENT  FigSeg           - - (%InfoUnitPool; | %Table;)* >
<!ATTLIST  FigSeg
    InfoMaster       NAME         #FIXED    "FigureSegment"
    %Common.Atts;
>
```

Figure (exhibits)

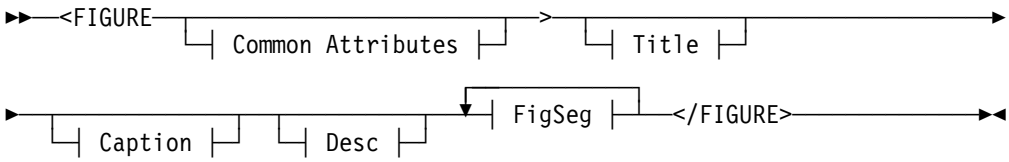
Purpose

The Figure element contains and identifies exhibits, such as images, examples, formulas, and tables.

Valid Contexts

Wherever paragraph- or phrase-level elements are valid.

Syntax



Attributes and Contained Elements

FigSeg

Figures are divided into one or more segments. Use multiple segments when the figure content needs to be broken into smaller parts, for example, when you have a long example. Presentation systems can use the segments to help determine the best presentation for the figure. For example, in a printed presentation, the system can try prevent segments from splitting across pages. Segments can also be used to provide hyperlink anchors within the content of a figure. For example, in a long program example, you might make each subroutine a separate segment and give each segment an ID so you can then link to subroutines directly.

ParaLevelElements

You can place almost anything within a figure except divisions.

Usage

Use figures to contain exhibit elements, such as graphics, examples, formulas, and tables. The figure serves to contain the exhibits and associate a title, caption, or description with them.

You can define different classes of figures in order to further define the content of the figure. For example, in a document that has both code samples and samples of reports, both of which are contained within example elements, you might define two figure classes, codesample and report so that you can distinquish between

code samples and reports. You could then, for example, generate two different figure lists, one for each class, or number the figures differently. Use ClassDef elements to define figure classes.

Use figure segments to break long exhibit elements into smaller chunks to enable breaking of figures at logical points. In a code sample, for example, you might use one figure segment for each subroutine, ensuring that no subroutines will be broken in the middle.

IBM ID Doc defines the figure class 'table', which identifies figures that contain table exibits. In the default style, figures of class table can be listed and numbered separately from other figures and will have their captions and descriptions presented at the top of the figure rather than at the bottom.

Examples

Simple example showing a piece of artwork within a figure:

```
<figure id=figure.a>
<figcap>A Sample Figure
<multimediaobject>
<objectcontainer name=squaredance>
<textalt>
<p>The graphics contains a box and a circle, with the
circle overlapping the lower right quandrant of the box.
</textalt>
</multimediaobject>
</figure>
```

Example showing the use of FigSeg to break up a long example.

```
<figure id=figure.b class=codesample>
<caption>A Long Code Sample
<desc>This code sample is very very long.  Each segment
contains a example which contains a reference an a NDATA
entity containing the live code.
<figseg>
<xmp>
&codesamp1;
</xmp>
</figseg>
<figseg>
<xmp>
&codesamp2;
</xmp>
</figseg>
  ⋮
<figseg>
<xmp>
&codesamp10;
</xmp>
</figseg>
</figure>
```

SGML Definition

```
<!ELEMENT  Figure           - -
                      (Title?, Caption?, Desc?,
                       (FigSeg* | (%InfoUnitPool; | %Table;)*)
                      )
>
<!ATTLIST  Figure
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Display"
>
```

FigureList (list of figures)

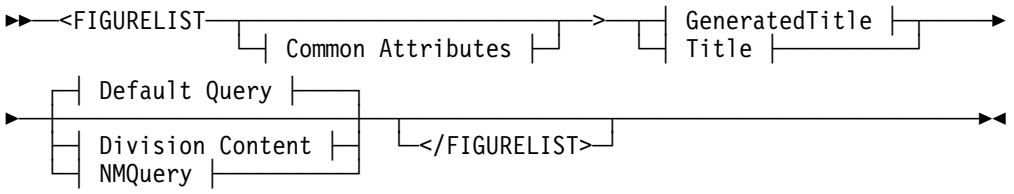
Purpose

FigureList contains either an explicit list of figures or a NMQuery element that resolves to a list of figures.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the FigureList division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

NMQuery
Division Content

The content of FigureList is either a NMQuery element, the content as for other divisions, or nothing.

If you specify no content, then the FigureList is defined to represent an application-defined query, where the actual query is defined either in the application implementation or via the application-specific style mechanism.

Default Query

The suggested default query for FigureList is to list all Figure elements for which a caption has been specified, listed in either the order they occur in the presentation, where there is a defined presentation order, or in the order they occur in the source, where there is no defined presentation order, as in the case of an unstructured hypertext database.

Usage

Use FigureList to contain or create a list of figures. The simplest use of FigureList is as follows:

```
<FigureList><GeneratedTitle>
```

This usage causes the presented figure list to be created using the default query, which is defined by the processing application. The figure list title is also defined by the processing application.

In the case where you need to explicitly define the content of the figure list, do so using the normal elements valid within all divisions:

```
<FigureList><GeneratedTitle>
<DivBody>
<p>See the following locations for interesting graphics:
<ul>
<li><xref refid=figure1>
<li><xref refid=figure2>
</ul>
</DivBody>
</FigureList>
```

Examples

```
<frontmatter>
<FigureList><generatedtitle>
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  FigureList       - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool)*)
                                )
>
<!ATTLIST  FigureList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>
```

FN (footnote)

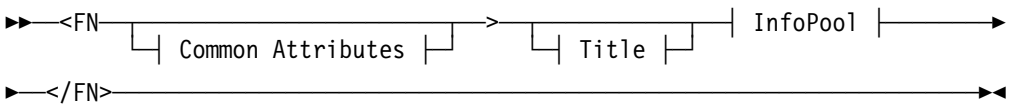
Purpose

FN is used to annotate text. FN is valid wherever DataPool or InfoUnitPool is valid.

Valid Contexts

InfoPool.

Syntax



Attributes and Contained Elements

Title

The title of the footnote. The title is optional. Depending on the hyperlink style in effect, the title may be used as the hyperlink button or cross-reference text for the footnote.

Usage

Use FN to annotate text with notes that are not appropriate for inclusion in line or to indicate the source for facts or other material used in the text. Footnotes are associated with the content of the element containing the footnote. For example, a footnote within a paragraph is associated with the entire paragraph and in the default presentation style the reference to the footnote will be placed at the end of the paragraph.

When you want to associate a footnote with a specific part of a paragraph, you can either use a Phrase element to enclose the text to be associated with the footnote or use XRef to refer to a footnote that has been defined elsewhere. In the default processing style, footnotes that have an ID= value specified for them are only presented when linked to explicitly by XRef or by the Annotations= attribute of an element.

Examples

Applying a footnote to text:

```
<p>This is text before a footnote.
<phrase>
This is data that
<fn>This is the footnote</fn>
the footnote applies to.
</phrase>
This text follows the footnote.
```

Footnotes can be refered to using the XRef element:

```
<p>This is an important statement.<xref refid=importfn>
</p>
<fn id=importfn>
No really, it's important.
</fn>
```

Footnotes can also be refered to using the Annotations= attribute:

Fragment (syntax fragment)

```
<p><phrase annotations=importfn>This is an important
statement.</phrase>
</p>
```

SGML Definition

```
<!ELEMENT  FN              - - (Title?, FNBody)          -- FootNote -->
<!ATTLIST  FN
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!ELEMENT  FNBody          O O (%InfoPool;)*       -- Foot Note Body -->
<!ATTLIST  FNBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ParagraphBody"
>
```

Fragment (syntax fragment)

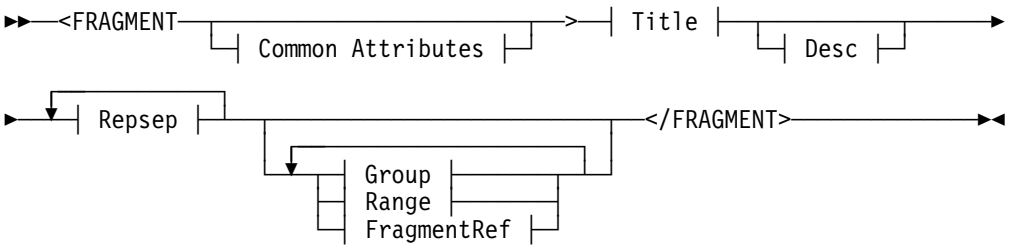
Purpose

Contains a labeled subpart of a syntax definition.

Valid Contexts

Syntax, SyntaxBlock.

Syntax



Attributes and Contained Elements

Title
The title of the fragment.

RepSep
Defines any complex repeat separators used by groups within the fragment.

Group
Syntax definitions are composed of one or more groups, which then contain and organize the syntactic elements themselves. As a matter of practice, each

logical entity within the statement should be contained within its own Group. See “Group (syntax grouping)” on page 135 for details on groups.

Range
Ranges define a range of values, usually by reference to a set. Ranges are analogous to groups except that their content is defined algorithmically in terms of set elements, rather than explicitly with syntactic elements. See “Range (syntactic range)” on page 278 for details on ranges.

FragmentRef
A reference to a syntax fragment. The reference is a logical, not a literal reference, so the fragment refered to does not actually have to exist. As for any element, the common LinkEnds= attribute can be used to make the reference literal. Presentation systems can also use the fragment name to create associative links.

Usage

Syntax fragments are in all ways the same as full syntax definitions except that they do not contain sets or syntax blocks. Use syntax fragments to organize subparts of a large syntax definition that either appear multiple times or are recursively defined, or parts that are too complicated to appear in place. Fragments are refered to with the FragmentRef element.

Examples

Recursive definition using a fragment:

```
<syntax>Group
  <fragment>Group Element
     <group>Group
        <fragmentref>Group Element
        <fragmentref>Range Element
        <fragmentref>Syntactic Elements
     </group>
  </fragment>
</syntax>
```

SGML Definition

```
<!ELEMENT  Fragment         - -
                     (Title, Desc?, RepSep*,
                      (Group | Range | FragmentRef | SyntaxNote)*)
>
<!ATTLIST  Fragment
    %Common.Atts;
>
```

FragmentRef (syntax fragment reference)

FragmentRef (syntax fragment reference)

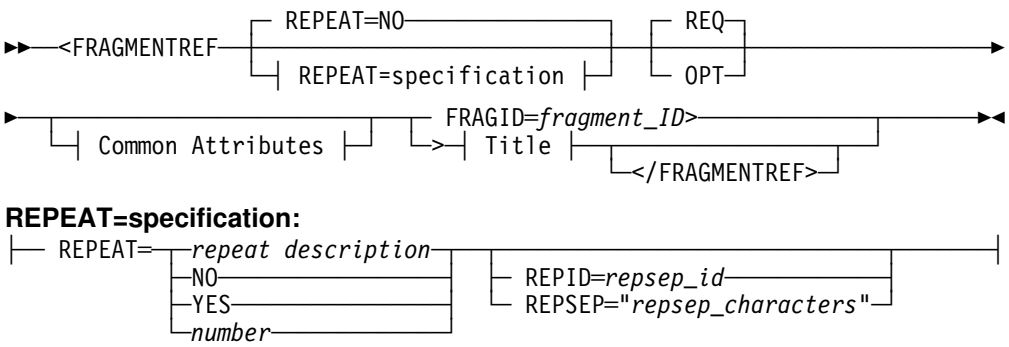
Purpose

Provides a logical reference to a syntax definition fragment.

Valid Contexts

Fragment, Group, RepSep, Syntax, SyntaxBlock, SyntaxPhrase.

Syntax



Attributes and Contained Elements

REPEAT=*repeat*

Specifies whether or not the fragment is repeatable and if so, how many times. The specification can be the keywords **YES** or **NO**, a single integer, or a prose specification of the stop rule, for example, “choose one or more items. Do not repeat any items.”.

REPID=*repsep_id*

The ID of a REPSEP element to be used as the repeat separator. The repeat separator can also be specified with the REPSEP= attribute.

REPSEP=*separator_value*

Specifies a simple repeat separator. This value is treated as if it were specified as a SEP element. REPSEP= and REPID= are mutually exclusive.

REQ | OPT

Indicates whether or not the fragment is optional. REQ (required) is the default

FRAGID=*fragment_ID*

Refers to a Fragment element. When FragID= is specified, is is an error to specify any content or the FragmentRef end tag.

Title

The title of the fragment reference. The fragment reference title need not match an actual fragment elsewhere in the syntax definition or document, but if it does, online systems may provide associative links between fragment references and fragments. Explicit hyperlinks can be defined between fragment references and fragments using the FragID= attribute.

Usage

Use fragment references to create symbolic references to subparts of a syntax definition or to abstract constructs that are not explicitly defined. For example, you can reduce the complexity of a definition by replacing complex subparts with fragment references with meaningful titles. You can also use fragment references to define recursive constructs or to refer to abstract constructs that are not explicitly defined.

There is no requirement that the fragment being refered to actually exist. If the fragment being refered to does exist, you can refer to it explicitly with the FragID= attribute in order to pick up its title automatically.

Examples

```
<syntax>Group
  <fragment>Group Element
     <group>Group
        <fragmentref>Group Element
        <fragmentref>Range Element
        <fragmentref>Syntactic Elements
     </group>
  </fragment>
</syntax>
```

SGML Definition

```
<!ELEMENT  FragmentRef          - O (Title)     -- Fragment Reference -->
<!ATTLIST  FragmentRef
    FragID           NAME         #CONREF      -- Must be Fragment ID --
    Repeat           CDATA        #IMPLIED           -- Default is NO --
    RepID            NAME         #IMPLIED       -- Must be REPSEP ID --
    RepSep           CDATA        #IMPLIED  -- Repeat separator chars --
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                          "REPEAT ('YES' | 'NO' | number | rcdata)
                           &Common.LexModel;"
    RefType          CDATA        #FIXED
                          "RepID RepSep FragID Fragment &Common.RefType;"
    TargetType       CDATA        #FIXED    "Fragment"
    InfoMaster       NAME         #FIXED    "SymbolicReference"
>
```

Frontmatter

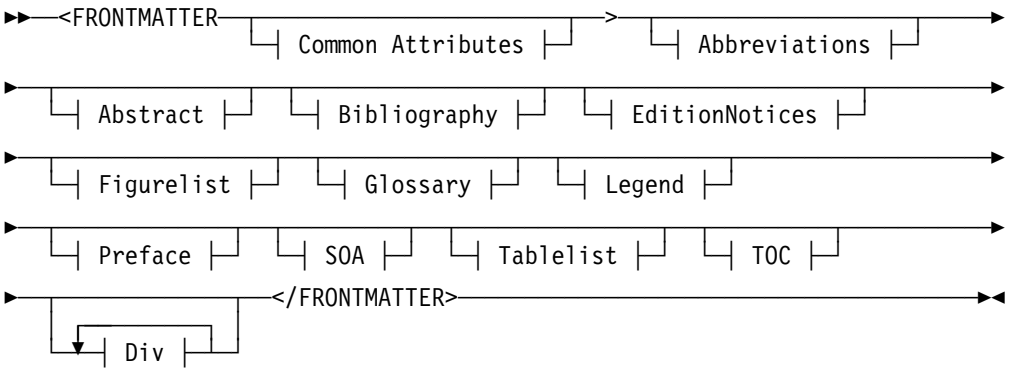
Purpose

Contains frontmatter elements such as Preface.

Valid Contexts

IBMIDDoc between Prolog and Body.

Syntax



Attributes and Contained Elements

Abbreviations
Contains a list of abbreviations.

Abstract
Contains the document abstract.

Bibliography
Contains the document bibliography. The content of Bibliography is normally one or more BibList elements.

EditionNotices
Contains any edition notices.

Figurelist
Contains a list of the figures included in the document.

Glossary
Contains the document glossary. The content of Glossary is normally one or more GL elements.

Legend
Contains the document legend, normally a description of symbols used throughout the document.

Preface
Contains the document preface.

SOA
Contains the document summary of amendments. The content of SOA is normally one or more MarkList elements.

Tablelist
Contains a list of the tables included in the document.

TOC
Produces a table of contents.

Div
Division elements with any valid division content.

Usage

Use Frontmatter to contain any frontmatter elements for the document. The FrontMatter specialized divisions are valid only within Frontmatter, and, in some cases, Backmatter. Use normal Div elements to contain information not accounted for by the specialized Frontmatter divisions.

Examples

```
<IBMIDDOC>
<Prolog>
  ⋮
</Prolog>
<FrontMatter>
  ⋮
</FrontMatter>
<Body>
  ⋮
</IBMIDDOC>
```

SGML Definition

```
<!-- '|' should be '&' and only Div allowed multiple times
     but tools don't properly support this *yet* -->
<!ENTITY % Frontmatter
    "(Abbreviations? | Abstract? | Bibliography? |
      EditionNotices? |
      FigureList? |
      Glossary? | Legend? | Preface? | SOA? | TableList? | TOC? |
      (%Safety;)? |
      Div
     )*"
>

<!ELEMENT  Frontmatter      - O (%Frontmatter;) >
<!ATTLIST  Frontmatter
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

GL (glossary list)

Purpose

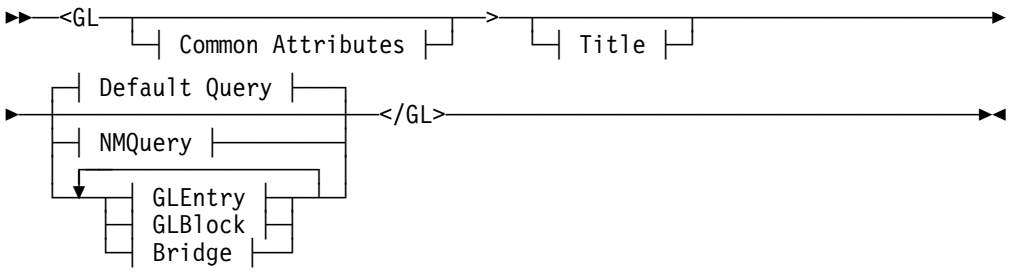
Contains glossary entries.

GL (glossary list)

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

Title
The title of the glossary list. When a title is specified, the Title start tag must be specified.

NMQuery
Default Query
Contains a query that defines the contents of the glossary list.

If neither NMQuery nor any other content elements are specified, the content of the GL is defined by the default query, specified in the active style specification.

The suggested default query is to retrieve all GLEntry elements within the document, sorted in order by the text of their terms or SortKey= values.

See “NMQuery (named query)” on page 225 for more information on using named queries.

GLEntry
A single glossary entry, which defines a single term. The GLEntry element contains two elements, DefinedTerm and one or more GDs (glossary description).

GLBlock
Contains a group of GLEntry elements in order to organize them for some author-defined purpose.

Bridge
A Bridge element used to provide a transition or connection between GLBlocks or GLEntries.

Usage

A glossary list contains one or more glossary entries (GLentry), which in turn contain a term and one or more definitions. Entries can be organized within a glossary list using glossary block (GLBlock) elements. Bridge elements can also be used to create transitions or connections between blocks of entries. Glossary lists have an optional title.

Glossary entries can also be used within normal element content to be collected automatically, or placed within document or division prologs when the terms apply to an entire document or a to specific division.

Unlike definition list entries, glossary entries can associate multiple definitions with a single term.

The content of a glossary list can also be defined by using a query. The query can either be the default glossary list query defined by a particular IBM ID Doc implementation or possibly in a style specification, or it can be an explicit query contained in or refered to from a NMQuery element.

Glossary lists are normally contained by a Glossary division in the front or backmatter, but they can occur anywhere paragraph-level elements are valid.

Examples

```
<gl>
<glentry>term
<gd>First definition.
<gd>Second definition.
<glentry>term
<gd>Definition
</gl>
```

SGML Definition

```
<!ELEMENT  GL              - - (Title?,
                                (nmquery |
                                 (GLEntry | GLBlock | Bridge)*
                                ))
                                                    -- Glossary List --
>
<!ATTLIST  GL
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefiningList"
>
```

GLBlock (glossary list block)

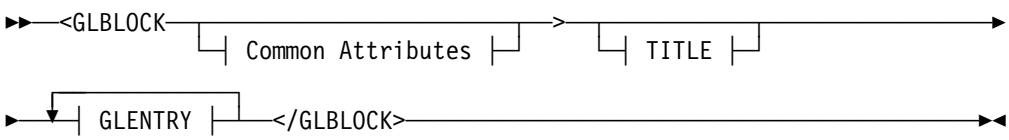
Purpose

Organizes glossary list entries within a glossary list.

Valid Contexts

GL.

Syntax



Attributes and Contained Elements

GLEntry

A single glossary entry, which defines a single term. The GLEntry element contains two elements, DefinedTerm and one or more GDs (glossary description).

Usage

Use GLBlock to organize DL entries arbitrarily. For example, you can use GLBlock to create logical subdivisions within a long glossary list, or use two GLBlocks with a bridge between them to highlight some relationship between groups of entries.

Examples

```
<gl>
<glblock>Ant-Eating Animals
<glentry>aardvark
<gd>...
<glentry>armadillo
<gd>...
</glblock>
<bridge><title>How Ant-Eating and Horse-Like Animals are Similar
<p>Ant-eating animals and horse-like animals
both tend to be plains-dwellers.
</bridge>
<glblock>Horse-Like Animals
<glentry>zebra
<gd>...
</gl>
```

SGML Definition

```
<!ELEMENT  GLBlock          - O (Title?, (GLEntry | Bridge)+)
                                    -- Glossary List Block of Entries --
>
<!ATTLIST  GLBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionBlock"
>
```

GLEntry (glossary list entry)

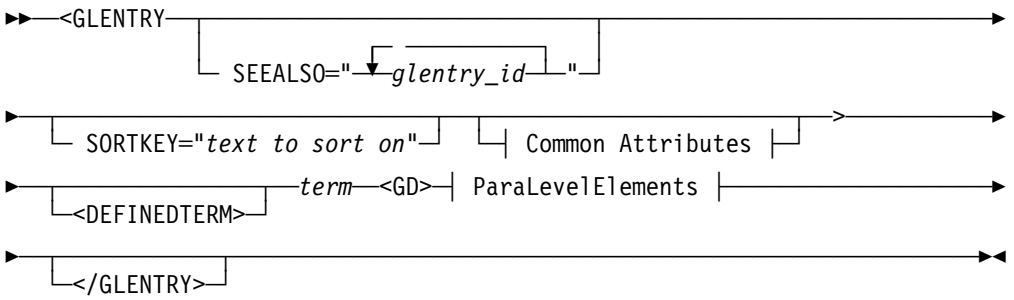
Purpose

Contains a single glossary term and its definition.

Valid Contexts

GL, GLBlock, PhraseLevelElements.

Syntax



Attributes and Contained Elements

SEEALSO=*glentry_id*
Contains references to other glossary entries related to this entry.

SORTKEY=*sortkey*
SORTKEY= defines the text characters to be used when sorting the glossary entry. The collating sequence used to do the sorting is a function of the current language, which may be determined by the container that holds the glossary entry or may be determined by the processing application. In most applications, the collating sequence will be the one defined for the language of the glossary entry itself.

term
The term being defined. Within DefinedTerm, hyphens (‐) define allowed hyphentation points within the term. The entity &reqhyph; defines hyphens that are part of the word and not simply hyphenation points. Under normal circumstances, the hyphens that define hyphenation points are not presented.

When the DefinedTerm element is given an ID, it can be used by reference from wherever PhraseLevelElements is valid by specifying a DefinedTerm element with a ContentRef= attribute.

Usage

The DefinedTerm start tag can be omitted. When the GLEntry element is given an ID, its content can be used by reference from within another glossary list or from within general content via a GLEntry element using the ContentRef= attribute.

Glossary entries are part of PhraseLevelElements, and thus can be used outside the context of a glossary list anywhere PhraseLevelElements elements are allowed. The default behavior when glossary entries are used this way is for the term to be presented where the entry is specified and for the term and definition to be collected for later use in a generated or dynamic glossary.

Examples

```
<gl>
<glentry><definedterm id=apple>ap-ple
<gd>A fruit of the apple tree.
</glentry>
</gl>
  ⋮
<p>An <DefinedTerm contentref=apple> a day keeps the doctor away.
```

SGML Definition

```
<!ELEMENT  GLEntry          - O (DefinedTerm, GD+)
                                          -- Glossary List Entry --
>
<!ATTLIST  GLEntry
    SeeAlso          NAMES        #IMPLIED
    SortText         CDATA        #IMPLIED
    %Common.Atts.NoRefType;

    InfoMaster       NAME         #FIXED    "Term"
    RefType          CDATA        #FIXED
                          "SeeAlso GLEntry &Common.RefType;"
>
<!ELEMENT  GD               - O (%InfoPool;)*
                                    -- Glossary List Term Definition --
>
<!ATTLIST  GD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermDefinition"
>
```

Glossary

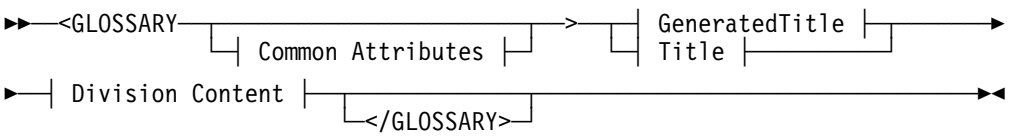
Purpose

Glossary is intended to contain a glossary of terms used in the document.

Valid Contexts

Frontmatter or Backmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title
You must specify either a GeneratedTitle element or a Title element containing the title for the Glossary division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

Division Content
Glossary has the same content rules as Div elements.

Usage

Use Glossary to contain glossary lists for a single document. The glossary lists themselves may generate their content via query.

Examples

```
<frontmatter>
<Glossary><generatedtitle>
<gl>
<glentry>apple
<gd>A fruit, much like a pear
</glentry>
</gl>
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  Glossary         - O (%SpecialDiv;) >
<!ATTLIST  Glossary
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

Group (syntax grouping)

Purpose

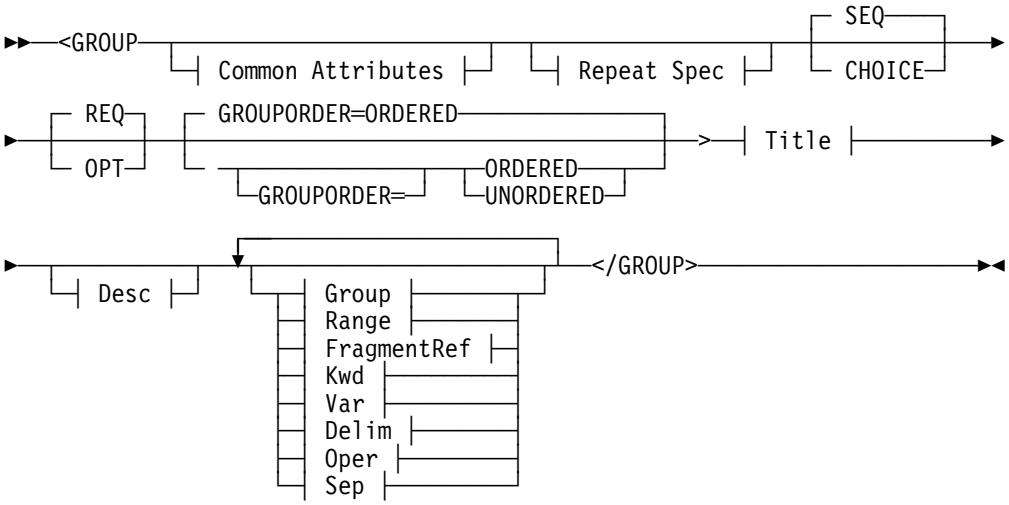
Group organizes and further defines the syntactic role of syntactic elements within a syntax definition.

Valid Contexts

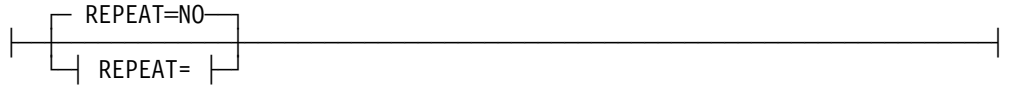
Fragment, Group, RepSep, Syntax, SyntaxBlock, SyntaxPhrase.

Syntax

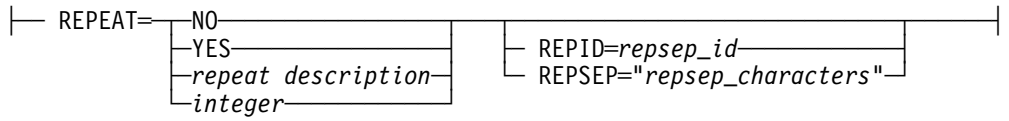
Group (syntax grouping)



Repeat Specification:



REPEAT=:



Attributes and Contained Elements

REPEAT=*repeat*

Specifies whether or not the group is repeatable and if so, how many times. The specification can be:

YES

Indicates that the group is repeatable with no limit.

NO

Indicates that the group is not repeatable (the default).

integer

Indicates the number of times the group can be repeated.

repeat description

A prose specification of the stop rule, for example, "choose one or more items. Do not repeat any items.".

REPID=*repsep_id*

The ID of a REPSEP element to be used as the repeat separator. The repeat separator can also be specified with the REPSEP= attribute.

REPSEP=*separator_value*

Specifies a simple repeat separator. This value is treated as if it were specified as a SEP element. REPSEP= and REPID= are mutually exclusive.

SEQ | CHOICE

Indicates whether or not the group is a set of choices. Groups are by default sequential (SEQ).

REQ | OPT
Indicates whether or not the group is optional. Groups are required by default.

ORDERED | UNORDERED
Indicates whether or not the order of elements within the group is important. The default is **ORDERED**.

Title
The title of the group. The group title is used as the fragment name should the group need to be turned into a fragment in the event the group is too wide to fit in the presentation space.

Desc
Describes the content of the group. Desc can be used to keep the description of a parameter with its definition rather than putting it in a separate parameter list.

Group
Groups can themselves contain groups. The nesting depth is limited only by the SGML TAGLVL value as defined in the SGML declaration.

Range
Defines a choice among a range of values as defined in a Set element.

FragmentRef
A reference to a syntax fragment.

Kwd
Contains a literal keyword.

Var
Contains a syntactic variable.

Delim
Contains a syntactic delimiter. Delimiters are normally containing in that they have a start and an end, as opposed to separators, which serve to separate subelements within a statement. Typical delimiters are quotation marks and parentheses.

Sep
Contains a syntactic separator. Separators separate one subelement of a statement from another. Typical separators are significant blanks and commas.

Oper
Contains a syntactic operator. Operators normally connect two elements together and specify an operation performed on one or both elements. Typical operators are the equals sign (=) and the plus sign (+). Operators may also be unary, for example, the negation operator.

Usage

Use groups to organize syntactic elements into logical groupings, typically one logical subelement per group; define sets of choices; indicate optionality; and indicate repeatability. As a rule of thumb, most logical subelements consist of a separator followed by a parameter, typically a keyword or variable. In non-positional languages (for example, SGML attribute specifications), separators are always associated with parameters and always precede the parameter.

Use the Desc element within a Group to contain the descriptive information for the group. For example, if a given group contains the definition of a single parameter, you could use the Desc element to describe that parameter. It would then be possible for the processing system to automatically generate a parameter list where the parameter is the content of the Group and the parameter description is the content of the Desc element.

Examples

```
<group>Group Element
  <desc>Use the Group element to ...
  <delim start>&stago;
  <kwd>GROUP
  <delim start>&tagc;
    <fragmentref repeat="yes" opt>Group Elements
  <delim start>&etago;
  <kwd>GROUP
  <delim start>&tagc;
</group>
```

SGML Definition

```
<!ENTITY % GroupElem
    "Group | Range | FragmentRef | Kwd | Var | Delim | Oper | Sep |
     SyntaxNote"
>

<!ELEMENT  Group            - - (Title?, Desc?, (%GroupElem;)*) >
<!ATTLIST  Group
    Repeat           CDATA        #IMPLIED
    RepID            NAME         #IMPLIED       -- Must be REPSEP ID --
    RepSep           CDATA        #IMPLIED  -- Repeat separator chars --
    OptReq           (Opt | Req| Def)  Req
    ChoiceSeq        (Seq | Choice)
                                  Seq
    GroupOrder       (Ordered | Unordered)
                                  Ordered
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                          "REPEAT ('YES' | 'NO' | NUMBER | RCDATA)
                           &Common.LexModel;"
    RefType          CDATA        #FIXED
                          "RepID RepSep &Common.RefType;"
>
```

Hex (hexadecimal number)

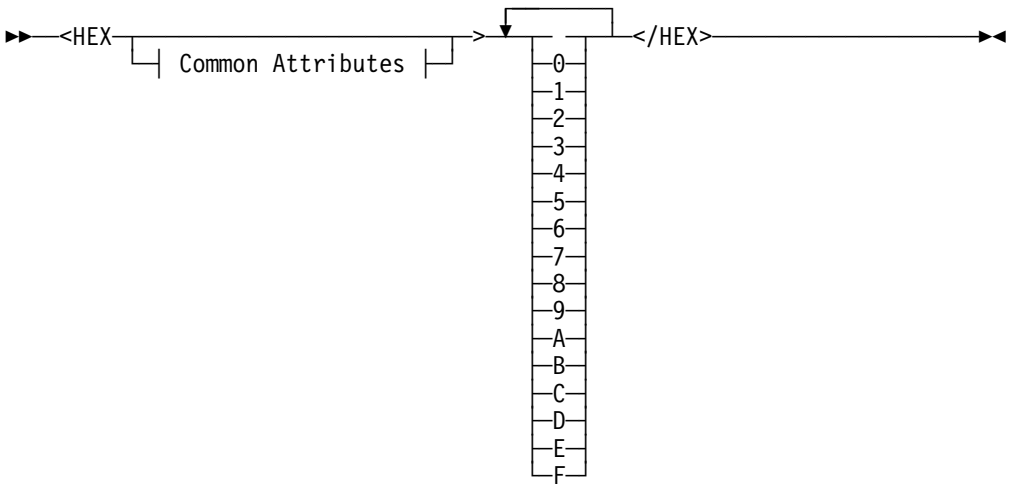
Purpose

The Hex element identifies hexidecimal data. Hex is valid wherever PhraseLevelElements is valid.

Valid Contexts

PhraseLevelElements.

Syntax



Usage

Use the Hex element to identify hexidecimal data. Hexidecimal data is data that is encoded in a base 16 numbering system. The first 6 English alphabetic characters are used to represent the decimal numbers 10 thru 15 in this base 16 numbering system. Its content is the sixteen hexidecimal digits, '0' thru '9' and 'A' thru 'F', and the space character. Either the upper or lower case letters may be used. A space character in the content of this number represents a visual separator within a single hexidecimal number (not a separator between multiple numbers).

Examples

```
<bin>11000001</bin> = <decimal>193</decimal> = <hex>C1<hex>
```

SGML Definition

```
<!ELEMENT  Hex              - - (#PCDATA)  -- hexadecimal data phrase -->
<!ATTLIST  Hex
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "HexidecimalData"
    LexModel         CDATA        #FIXED
                          "#CONTENT
                              (digits | 'A'|'B'|'C'|'D'|'E'|'F' | ' ')*
                           &Common.LexModel;"
>
```

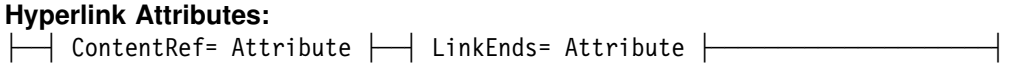
Hyperlink Attributes (hyperlink definitions)

Purpose

These attributes define the hyperlinking semantics provided by &bookie. and also provide an attribute to encode user-defined linking semantics. element.

Syntax

Hyperlink Attributes:



Attributes and Contained Elements

ContentRef= Attribute

Refers to another element of the same type in order to use its content. When ContentRef= is specified, it is an error to specify any content for the element or to specify the element's end tag.

LinkEnds= Attribute

Used to establish IBM ID Doc-defined hyperlinks and author-defined hyperlinks. See “LinkEnds, AnchorRoles, and HyTime Attributes (hyperlink target reference attributes)” on page 187 for details on using the LinkEnds= attribute.

Usage

Each of these attributes defines a specific type of hyperlink to another element for a specific purpose. See the description of each attribute for details on its use.

SGML Definition

```
<!ENTITY % Hyperlink.Attributes.BookMaster
   "Qualification    NAMES        #IMPLIED"
>

<!ENTITY % Hyperlink.AnchRoles.BookMaster
   "RevisionSpecification"
>

<!ENTITY % Hyperlink.AnchorRoles
   "Annotation | PropertySource | Alternative"
>

<!ENTITY % Hyperlink.Attributes
   "ContentAssociations
                     NAMES        #IMPLIED
    AnchorRoles      NAMES        #IMPLIED
    LinkEnds         NAMES        #IMPLIED
    %Hyperlink.Attributes.BookMaster;"
>
```

IBMBibRecord (IBM bibliographic record)

Purpose

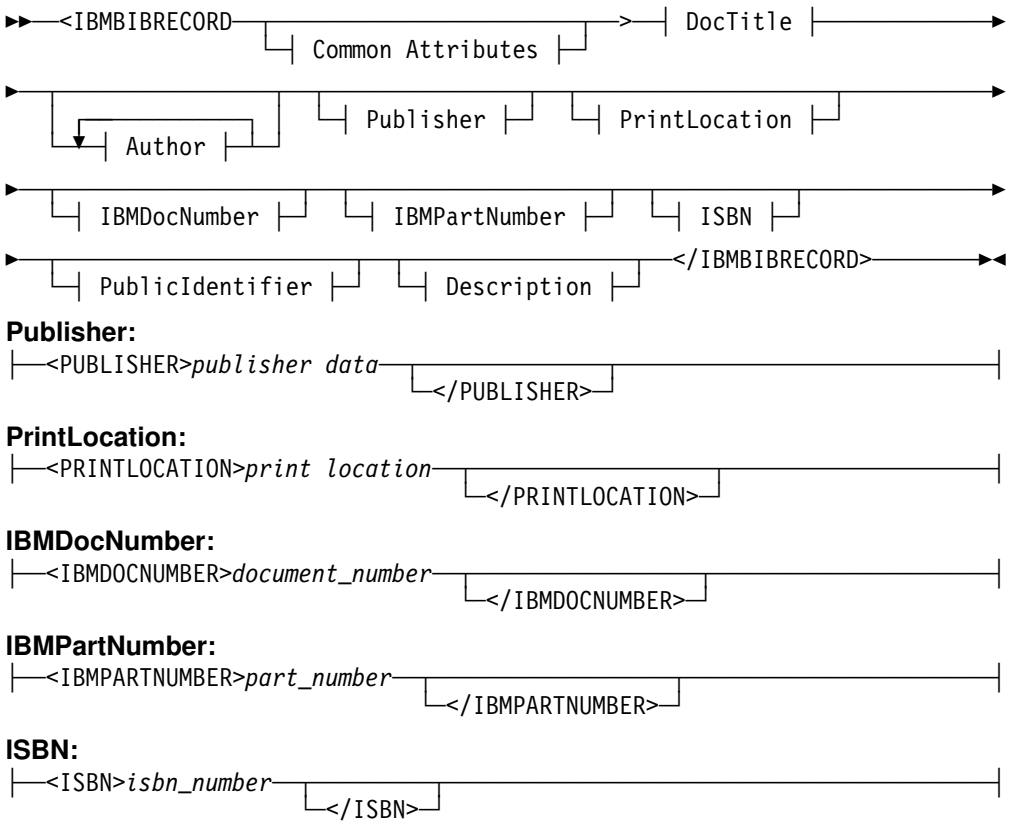
Contains an IBM-specific bibliographic record.

Bibliographic records structure and organize the information about documents. The bibliographic information can be used in a variety of ways, including generating bibliographies.

Valid Contexts

BibRecords, BibList, Citation, LongQuote, Quote, BibRecord, LibRecord, IBMLibRecord.

Syntax



Attributes and Contained Elements

DocTitle
The title of the document.

Author
One author of the document. You can specify multiple authors.

Publisher
Contains the name of the publisher followed by an optional address element.

PrintLocation
The country where the document is printed.

IBMDocNumber
The IBM document number assigned to the document described in this record.

IBMPartNumber
The IBM part number assigned to the document described in this record.

ISBN
The ISBN number assigned to the document described in this record.

PublicIdentifier
The SGML public identifier assigned to the document described in this record. This is the same public identifier used in entity declarations for the document described in this record.

Description
A Desc element containing a description of the document.

Usage

Use IBMBibRecord to define the bibliographic information about an IBM document.

IBMBibRecord elements can also be specified in a BibRecords element or object container and used by reference from within a document, for example, from Citation and Quote elements.

When Linkends= is used with the anchor role DescribedDocument, the IBMBibRecord element becomes the anchor of a link to the document described. When the unified name space option is in effect, the Linkends= value can be the name of the entity that represents the described document. When the unified name space option is not in effect, you must create Nameloc element to map the ID reference used in the Linkends= value to the entity name (see “Nameloc (named location)” on page 220 for details on using the Nameloc element).

The default hyperlink style is for the BibRecord and all references to it to be active hyperlinks to the document.

When IBMBibRecord is specified within Citation, Quote, and LongQuote, the IBMBibRecords are collected for use in a generated bibliography (see “BibList (bibliography entry list)” on page 56).

It is intended that the public identifier of the document entity be used by the presentation system to locate the actual document, but specific presentation systems may define application-specific data to be specified as the system identifier of the document entity if they do not support the use of public identifiers. The public identifier can be included in the IBMBibRecord itself as a way of keeping a document's formal public identifier definition with the rest of its bibliographic information. This could allow, for example, the automatic generation of entity declarations for documents described by BibRecord elements.

See Appendix A, “Proposed IBM Standard for Formal Public Identifiers” on page 332 for a description of the formal public identifier standard defined by the IBM Corporation for its internal and external use.

Examples

```
<IBMbibrecord id=bookieref>
 <DocTitle>
 <shorttitle>BookMaster Reference
 <Library>IBM ID Doc
 <topic>Language Reference
 </DocTitle>
 <author><name>Day<first>Don<middle>R.</name>
 </author>
 <author><name>Kimber<first>William<middle>E.</name>
 </author>
 <author><name>Temple<first>Mike</name>
 </author>
 <author><name>Wohler<first>Wayne<middle>L.</name>
 </author>
 <Publisher>IBM Corporation
  <address>
  </address>
 </publisher>
 <PrintLocation>USA
 <IBMDocNumber>SC32-5009-11
 <PublicIdentifier>+//ISBN 0-19-9999
 //PUB SC32-5009-11
 /IBM ID Doc Reference
 /drmacro@ralvm13.vnet.ibm.com
 /Publishing Solutions
 //EN
 <Desc>Defines the BookMaster language.
 BookMaster is an SGML language for creating printed and online
 technical information.
</IBMbibrecord>
```

SGML Definition

```
<!ELEMENT  IBMBibRecord     - O
                     (DocTitle, VolumeIdentifier?, RetrievalKey?,
                      Authors?,
                      Publisher?, PrintLocation?,
                      FileNumber?,
                      IBMDocNumber?, IBMPartNumber?, ISBN?,
                      PublicIdentifier?, Desc? )
                                        -- IBM Bibliographic Record --
>
<!ATTLIST  IBMBibRecord
    %LibRecord.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
    ApplicationClass NAME         #FIXED    "BibliographicRecord"
>

<!ELEMENT  Publisher        - O (EnterpriseName, Address?) >
<!ATTLIST  Publisher
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
             -- This element also conforms to the NameAndAddress form --
```

```
    ApplicationClass NAME         #FIXED    "Publisher"
>

<!ELEMENT  PrintLocation    - O (%DataPool;)* >
<!ATTLIST  PrintLocation
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "PrintLocation"
>

<!ELEMENT  IBMDocNumber     - O (#PCDATA)      -- IBM Document Number -->
<!ATTLIST  IBMDocNumber
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "IBMDocNumber"
    LexModel         CDATA        #FIXED
       '(("SC" | "GC" | "LY" | "ZZ"), digit, digit, "-", digit,
         digit, digit, digit, "-", digit, digit)
        &Common.LexModel;'
>

<!ELEMENT  PartNumber       - O (#PCDATA) >
<!ATTLIST  PartNumber
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "PartNumber"
>

<!ELEMENT  IBMPartNumber    - O (#PCDATA) >
<!ATTLIST  IBMPartNumber
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "IBMPartNumber"
    LexModel         CDATA        #FIXED
       '(digit, digit, digit, digit, digit, digit, digit)
        &Common.LexModel;'
>

<!ELEMENT  ISBN             - O (#PCDATA) >
<!ATTLIST  ISBN
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ISBN"
    LexModel         CDATA        #FIXED
       '(digit, "-", digit, digit, "-", digit, digit, digit, digit,
         digit, digit, ("-", digit)?)
        &Common.LexModel;'
>

<!ELEMENT  PublicIdentifier - O (#PCDATA) -- Formal Public Identifier -->
<!ATTLIST  PublicIdentifier
    %Common.Atts.NoLexMdl;
```

```
    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "PublicIdentifier"
    LexModel         CDATA        #FIXED
       '(("-" | "+"), "//", owneridentifier, "//", publictextidentifier,
        "//", languagecode)
        &Common.LexModel;'
>
```

IBMIDDOC (IBM-specific product documentation)

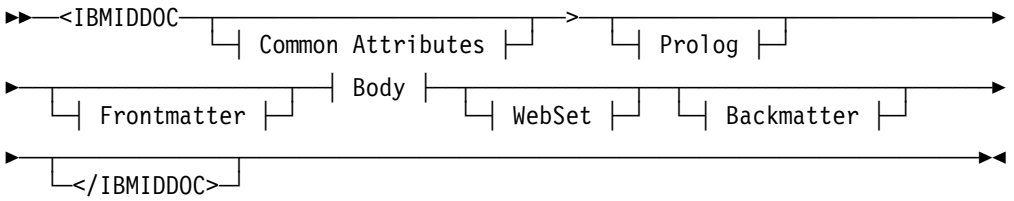
Purpose

Contains IBM product information. The IBM ID Doc document type is used within IBM to create information deliverables for IBM products. This document type conforms to the IBM InfoMaster Architecture and the HyTime standard (ISO/IEC 10744).

Valid Contexts

Must occur in same entity (file) as the IBMIDDoc document type declaration and must be highest-level element in the document.

Syntax



Attributes and Contained Elements

Prolog
Contains the document-level metainformation, such as tracking information, revision information, and property definitions.

Frontmatter
Contains specialized sections such as PREFACE and SOA.

Body
Contains the body of the document, organized into divisions by Div elements or modular information elements.

WebSet
Contains one or more Web elements.

Backmatter
Contains the backmatter elements such as glossary, and bibliography.

IBMIDDOC (IBM-specific product documentation)

Usage

The IBMIDDOC element contains an entire IBMIDDOC document. Property attributes specified for the IBMIDDOC element will be inherited by all elements within the document.

An IBMIDDOC document is divided into five main elements: Prolog, Frontmatter, Body, WebSet, and Backmatter. Only the body element is required. The Prolog contains all the information that describes the document itself, such as the title, author, document numbers, property definitions, and so on. WebSet contains a set of hypertext webs. The other sections contain the content of the document organized into divisions, either by Div elements or by specialized divisions, such as Preface or Bibliography.

The minimum IBMIDDOC document is:

```
<div>Division Title
</div>
```

Examples

```
<IBMIDDOC sec="IBM Internal Use Only" language="USENGLISH"
                  id="edfl0mstv2r1">
  <prolog>
  </prolog>
  <frontmatter>
  </frontmatter>
  <body>
    <div>Introduction to this document
    </div>
  </body>
  <backmatter>
   <appendixes>
     <div>First Appendix
     </div>
   </appendixes>
</IBMIDDOC>
```

SGML Definition

```
<!ENTITY % DocumentType     "IBMIDDoc" >

<!ENTITY % DocumentElement
    "Prolog?, Frontmatter?, Body, WebSet?, Backmatter?"
>

<!ENTITY % Document.Atts
   '%Common.Atts.NoHyTime;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    HyTime           NAME         #FIXED    "HyDoc"     -- HyTime doc --
    unmspace         (unified | separate)
                                  #FIXED    "unified"'
>

<!ELEMENT  %DocumentType;   O O (%DocumentElement;)
```

```
                                                    -- IBM Software Document --
>
<!ATTLIST  %DocumentType;
    %Document.Atts;
>
```

IBMLibRecord (IBM document library definition)

Purpose

Contains an IBM-Specific library definition.

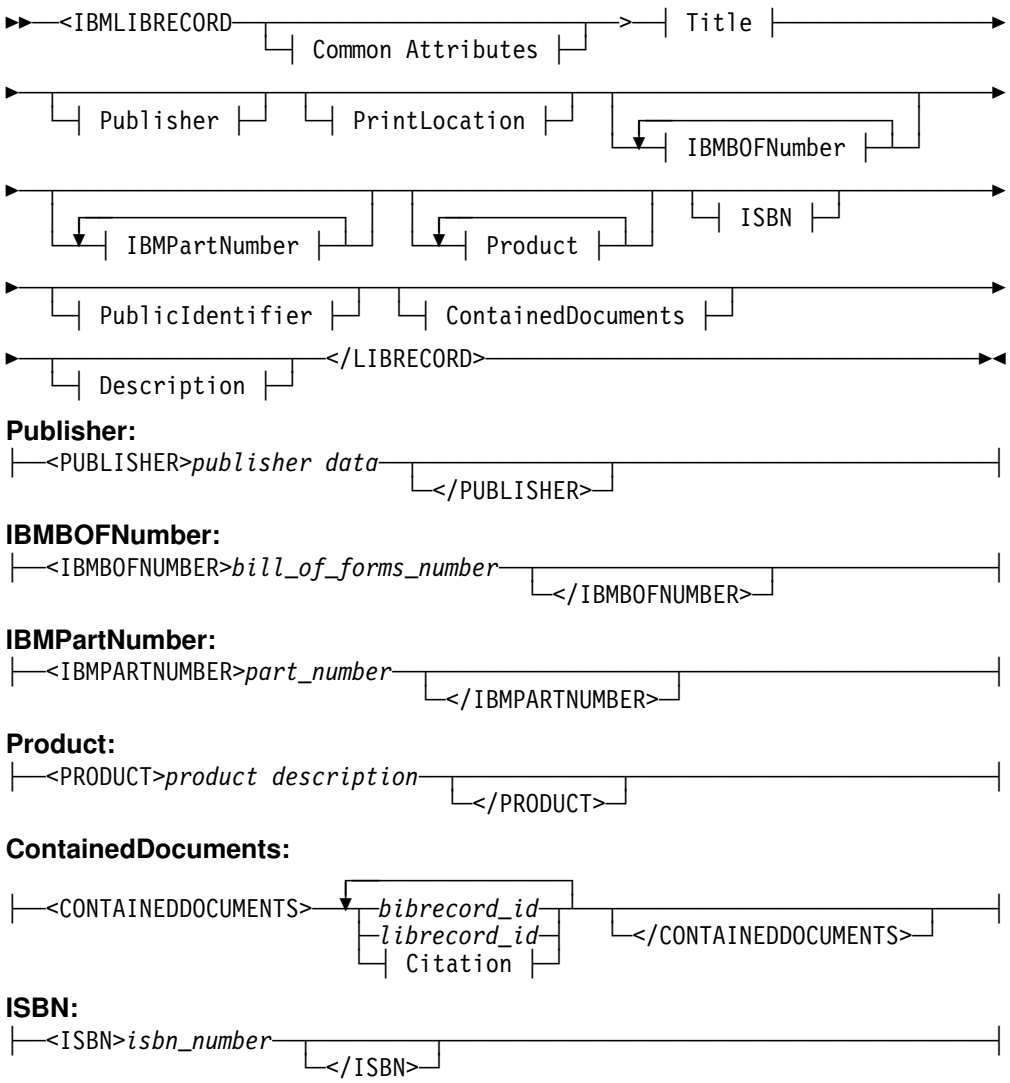
Library records structure and organize the information about libraries or collections of documents.

Valid Contexts

BibList, BibRecords, Citation, BibRecord, IBMBibRecord.

Syntax

IBMLibRecord (IBM document library definition)



Attributes and Contained Elements

Title
The title of the library.

Publisher
Contains the name of the publisher followed by an optional address element.

IBMBOFNumber
The IBM bill of forms (BOF) number assigned to this library.

IBMPartNumber
The IBM part number assigned to this library.

ISBN
The ISBN number assigned to this library.

PublicIdentifier
The SGML public identifier assigned to this library. This is the same public identifier used in entity declarations for the system object that represents this library.

ContainedDocuments
Defines the documents contained in this library and the default order for presenting the documents when the library definition is presented.

The contained documents can be identified directly using Citation elements, or indirectly by specifying BibRecord or LibRecord IDs.

The order the contained documents are specified in ContainedDocuments defines the default order for presenting the library contents.

Description
A Desc element containing a description of the library.

Usage

Use IBMLibRecord to define the bibliographic information about a library or other collection of IBM documents. You can use the common Class= attribute and the ClassDef element to define different classes of LibRecord to correspond to different classes of collection. For example, you may have product libraries that are themselves collected into larger libraries of libraries. You could define a class of 'CollectionKit' for libraries of libraries and then use IBMLibRecord elements to define the contents of a given collection kit.

IBMLibRecords can also be specified in a BibRecords section or object container and used by reference from within a document, for example, from Citation elements.

When Linkends= is used with the anchor role DescribedDocument, the IBMLibRecord element becomes the anchor of a link to the library described. When the unified name space option is in effect, the Linkends= value can be the name of the entity that represents the described library. When the unified name space option is not in effect, you must create Nameloc element to map the ID reference used in the Linkends= value to the entity name (see “Nameloc (named location)” on page 220 for details on using the Nameloc element).

The default hyperlink style is for the IBMLibRecord and all references to it to be active hyperlinks to the library.

When IBMLibRecords are specified within Citation. the IBMLibRecords are collected for use in a generated bibliography (see “BibList (bibliography entry list)” on page 56).

It is intended that the public identifier of the library entity be used by the presentation system to locate the actual system object, but specific presentation systems may define application-specific data to be specified as the system identifier of the library entity if they do not support the use of public identifiers. The public identifier can be included in the IBMLibRecord itself as a way of keeping a libraries's formal public identifier definition with the rest of its bibliographic information. This could allow, for example, the automatic generation of entity declarations for libraries described by IBMLibRecord elements.

See Appendix A, “Proposed IBM Standard for Formal Public Identifiers” on page 332 for a description of the formal public identifier standard defined by the IBM Corporation for its internal and external use.

Examples

```
<IBMlibrecord id=sgmlbookielib>
 <Title>
 <shorttitle>InfoMaster Library</shorttitle>
Publishing Solutions
IBM ID Doc Library
 <Publisher>IBM Corporation
  <address>
  </address>
 </publisher>
 <PrintLocation>USA
 <IBMBOFNumber>SBOF-6000-00
 <PublicIdentifier>+//ISBN 0-19-9999
 //LIB SBOF-6000-00
 /IBM ID Doc Library
 /drmacro@ralvm13.vnet.ibm.com
 /Publishing Solutions
 //EN
 <ContainedDocuments>bkmbookieug sbkmbookieref
 </ContainedDocuments>
 <Desc>Documentation for the IBM ID Doc language.
 IBM ID Doc is an SGML language for creating printed and online
 technical information.
</IBMlibrecord>
```

SGML Definition

```
<!ELEMENT  IBMLibRecord     - -
                      (Title, Publisher?, PrintLocation?,
                       IBMBOFNumber?, IBMPartNumber?,
                       Product*, ISBN?, PublicIdentifier?,
                       ContainedDocuments?, Desc?)
>
<!ATTLIST  IBMLibRecord
    %LibRecord.Atts;
>
```

IBMSafety (IBM safety notices)

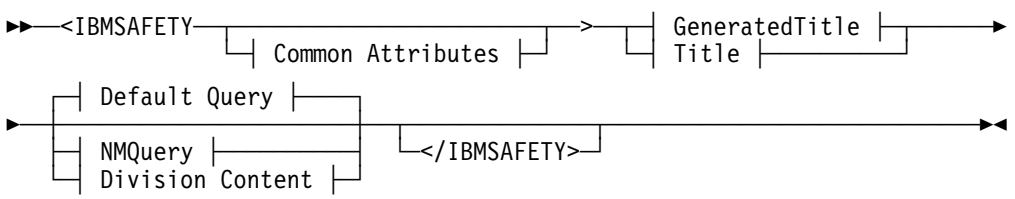
Purpose

IBMSafety is intended to contain IBM-specific safety notices.

Valid Contexts

Frontmatter

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the IBMSafety division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

NMQuery
Default Query

The content of the IBMSafety division can be generated by a query. If no division content is specified for the Safety element, then the default query will be used. Alternatively, you can specify an explicit query to generate the content.

The suggested default query is to locate all safety-related elements in the document.

Division Content

IBMSafety has the same content rules as Div elements.

Usage

Examples

```
<frontmatter>
<ibmsafety><generatedtitle>
&standardIBMSafety;<!-- Get boilerplate -->
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  IBMSafety        - O ((GeneratedTitle | Title),
                                  (nmquery | (%InfoUnitPool;)*)
                                 )
>
<!ATTLIST  IBMSafety
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>
```

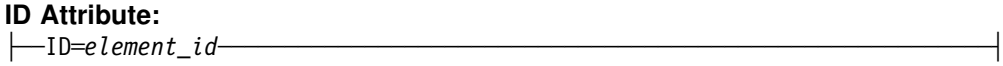
ID Attribute (element identifier)

Purpose

Uniquely identifies an element within some defined element ID scope. The ID scope is defined within a given application's document type definition. By default, IDs must be unique within a single document.

Syntax

ID Attribute:



Usage

All elements can take an ID= attribute. Any element that has an ID can be linked to, be identified as a retrieval alternative, or be used by reference via ContentRef=. IDs can be up to 32 characters long. IDs must start with an alphabetic character and can contain letters, numbers, or dashes (-).

Periods are valid in IDs, but are defined by IBM ID Doc to separate IDs within a qualified ID when ID qualification is supported. As a rule, you should not use periods as part of normal element IDs to avoid problems if elements are re-used in other contexts.

Examples

```
<p id=a-paragraph>This paragraph has an ID.
```

SGML Definition

```
    ID              NAME        #IMPLIED
                                -- Can be required on some elements --
```

Index

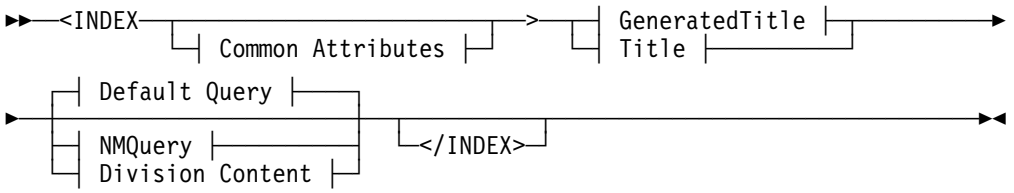
Purpose

The index element either contains an index, using whatever phrase- or paragraph-level elements are appropriate, or generates the index using a query, possibly defined in the active style specification.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the Index division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

Default Query
NMQuery

The query that defines what the presented content of the index will be. You can either specify a query with the NMQuery element or you can specify the content of the Index division explicitly following the rules for normal divisions.

If you specify neither a NMQuery nor any other division content, then the index is generated using the default index query.

The suggested default index query is to collect all index elements used or referred to within the document content and sort them alphabetically.

Division Content

Index has the same content rules as Div if you do not specify a NMQuery element.

Usage

Use Index to create or hold the index for the document. The normal use of Index is to create an index generated from the index entries specified or referred to from within the document content. However, you can also put any sort of content in Index that is valid within normal divisions.

For example, you might want to have the index division contain several subdivisions containing navigation aids, such as lists of functions, lists of commands, and lists of concepts in an software concepts document.

You can also define your own special queries to create a different index dynamically. However, this is a task normally expected to be performed by documentation specialists and may require interaction with your processing and presentation software.

Examples

```
<backmatter>
<index><generatedtitle>
<nmquery usefn="normal_index">
</frontmatter>
```

SGML Definition

```
<!ELEMENT  Index            - O (%SpecialDiv;) >
<!ATTLIST  Index
    Auto             (Auto)       #CONREF  -- Generated index --
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

InfoModule (information module)

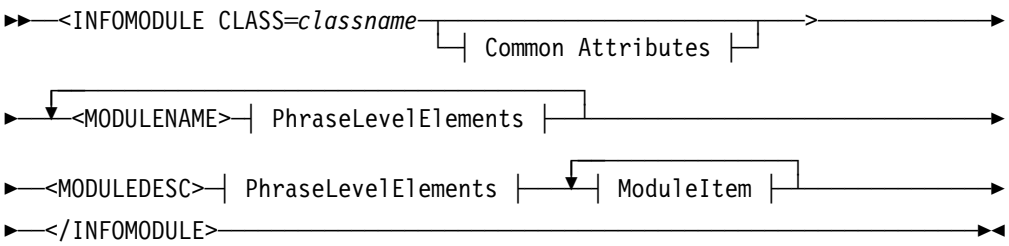
Purpose

An information module describes or defines a single object for a given modular information class as defined by the document author.

Valid Contexts

ModularInformation.

Syntax



Attributes and Contained Elements

CLASS=*classname*

Defines the class of this information module. Classes are defined with a ModularInfoDef element within the document or division prolog.

MODULENAME

The name of the modular information element, for example, a command name. You can specify multiple names for a give module. For example, a command may have several forms, such as the Unix grep command, which includes the grep, egrep, and fgrep versions.

MODULEDESC

A short description of the modular information element, intended to be part of the presented title.

MODULEITEM
One or more module description items.

Usage

Use modular information to create reference information for anything. You can define as many different modular information classes as you want. Modular information lets you create precise markup for repetitive information that exactly meets the needs of your documentation.

Examples

```
  ⋮
<Prolog>
  ⋮
<PropertyDefinitions>
  ⋮
<ModularInfoDef classname=commands>
 <desc>This class of modular information should be used to
 describe commands.
 </desc>
 <ModiItemDef classname=purpose>Purpose
  <desc>Contains a description of the purpose of the command.
 <ModiItemDef classname=syntax>Syntax
  <desc>Contains a description of the syntax of the command.
</ModularInfoDef>
  ⋮
</PropertyDefinitions>
  ⋮
</Prolog>
  ⋮
<ModularInformation class=command>
<InfoModule id=fubar>
<ModuleName>FUBAR command
<ModuleDesc>Slices, dices, makes julien fries
<ModuleItem class=purpose>
<p>FUBAR performs the slicing and dicing function.
<ModuleItem class=syntax>
<syntax>Fubar command
</syntax>
</InfoModule>
  ⋮
</ModularInformation>
```

SGML Definition

```
<!ELEMENT  InfoModule       - O (ModuleName+, ModuleDesc?, ModuleItem+)
                                          -- Information Module --
>
<!ATTLIST  InfoModule
    Class            NAME         #REQUIRED         -- author-defined --
    %Common.Atts.NoClass;

    InfoMaster       NAME         #FIXED    "InformationModule"
>
```

IRef (index entry reference)

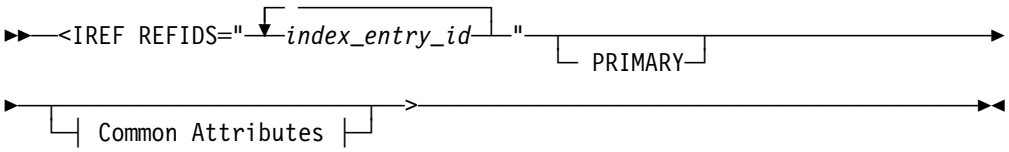
Purpose

The IRef element associates an element with an index entry by refering to an index entry defined elsewhere in the document, either in the content or in an IndexDefinitions element. IREF is valid wherever ParaLevelElements is valid.

Valid Contexts

PhraseLevelElements, ParaLevelElements.

Syntax



Attributes and Contained Elements

REFIDS=*index_entry_ids*
Refers to one or more index entries (I1, I2, or I3) to be associated with the element that contains this IRef.

PRIMARY
Indicates that this entry is a primary reference point.

Usage

Use explicit index entries when the indexing function provided by author-defined classes and phrase markup is not sufficient for your information. Index entries specified in the information content are normally used to generate a traditional index. When index entries are specified in an IndexDefinitions element within Prolog or DivProlog, they define index structures that can be used by reference from within the document content. Index entries within IndexDefinition will not appear in a generated index unless specifically referred to.

Index entries are associated with the element that directly contains them, regardless of where in that element they occur. For example, an index entry within a paragraph is associated with the entire paragraph. You can associate an index entry with a specific point within a paragraph by putting it within a phrase element.

You can define your own classes for index entries and assign special meaning and processing to those classes. For example, you could define a class called Search for index entries that are only to be used as search targets in online presentation systems. Classes are defined using the ClassDef element.

You can also use author-defined element classes to create indexes. See “ClassDef (element class definition)” on page 74 for more information.

Examples

```
<p><iref refids='aboutaconcept'>
This paragraph is about a concept.
```

SGML Definition

```
<!ELEMENT  IRef             - O EMPTY
                      -- Index reference.  Refers to an I1, I2, or I3. --
>
<!ATTLIST  IRef
    ID               NAME         #IMPLIED
    RefIDs           NAMES        #REQUIRED
    Primary          (Primary)    #IMPLIED          -- Primary entry? --
    %Common.Atts.Base;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED
                          "RefIDs (I1 | I2 | I3) &Common.RefType;"
>
```

I1 (first-level index entry)

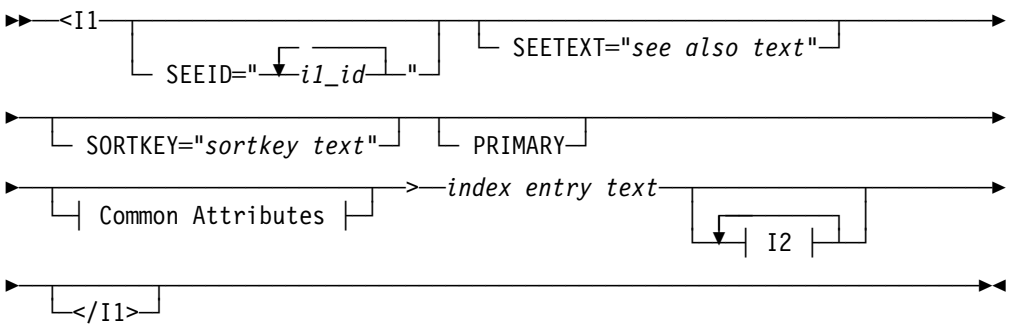
Purpose

The I1 element contains an index entry and any related secondary or tertiary (third-level) entries. An index entry is associated with the element that directly contains it. I1 is valid wherever PhraseLevelElements or ParaLevelElements is valid, or within an IndexDefinitions element within Prolog or DivProlog.

Valid Contexts

IndexDefinitions, PhraseLevelElements, ParaLevelElements.

Syntax



Attributes and Contained Elements

SEEID=*i1 ids*
Defines a cross reference to one or more other first-level index entries.

SEETEXT=*see also text*
Contains the text of a see or see also reference. SeeID= and SeeText= can both be specified, if desired.

SORTKEY=*sortkey text*
When specified, the SortKey= text is used to sort the entry, rather than the index entry text itself.

PRIMARY
Indicates that this entry is a primary reference point.

index entry text
The index term itself.

I2 One or more I2 elements.

Usage

Use explicit index entries when the indexing function provided by author-defined classes and phrase markup is not sufficient for your information. Index entries specified in the information content will be used to generate a traditional index. When index entries are specified in an IndexDefinitions element within Prolog or DivProlog, they define index structures that can be used by reference from within the document content. Index entries within IndexDefinition will not appear in a generated index unless specifically referred to.

You can link related index entries together with the SeeID= attribute.

Use the SortKey= attribute to define the character string to be used when sorting the index entries if it is different from the index entry text itself. For example, you may want a number to sort as though it were spelled out.

Index entries are associated with the element that directly contains them, regardless of where in that element they occur. For example, an index entry within a paragraph is associated with the entire paragraph. You can associate an index entry with a specific point within a paragraph by putting it within a phrase element.

You can define your own classes for index entries and assign special meaning and processing to those classes. For example, you could define a class called Search for index entries that are only to be used as search targets in online presentation systems. Classes are defined using the ClassDef element.

You can also use author-defined element classes to create indexes. See “ClassDef (element class definition)” on page 74 for more information.

Examples

```
<p><i1 seeid='relatedconcept'>concept</i1>
This paragraph is about a concept.
```

I1 can contain second level entries related to it:

```
<i1>concept<i2>about<i2>definition</i1>
```

SGML Definition

```
<!ELEMENT  IndexTerm        O O (%DataPool;)*
                                        -- index term PCDATA container --
>
<!ATTLIST  IndexTerm
    %Common.Atts;
>

<!ELEMENT  I1               - O (IndexTerm+, I2*)
                                     -- first-level index entry --
>
<!ATTLIST  I1
    SeeText          CDATA        #IMPLIED
    SeeID            NAMES        #IMPLIED
    SortKey          CDATA        #IMPLIED
    Primary          (Primary)    #IMPLIED          -- Primary entry? --
    %Common.Atts.NoRefType;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
    RefType          CDATA        #FIXED
                          "SeeAlso I1 &Common.RefType;"
>
```

I2 (second-level index entry)

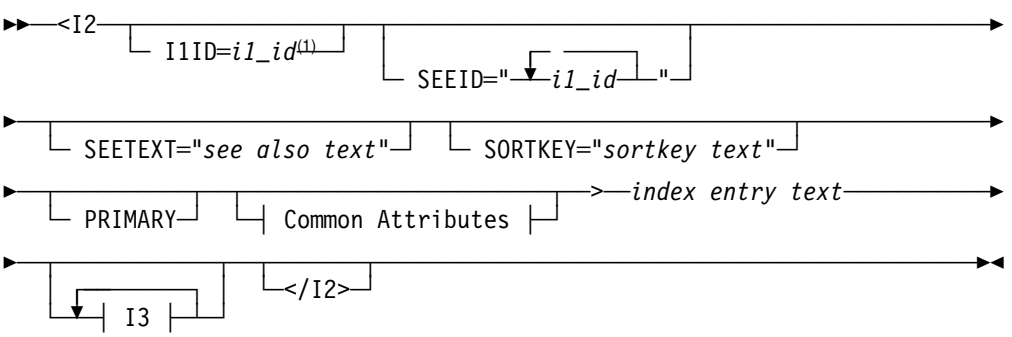
Purpose

The I2 element contains an index entry and any related tertiary (third-level) entries. An index entry is associated with the element that directly contains it. I2 is valid wherever PhraseLevelElements or ParaLevelElements is valid, or within an IndexDefinitions element within Prolog or DivProlog. I2 outside the context of I1 must use the I1ID= attribute to refer to an I1 element.

Valid Contexts

IndexDefinitions, i1, PhraseLevelElements, ParaLevelElements.

Syntax



Note:
[1] I1ID= is required when I2 occurs outside the context of I1.

Attributes and Contained Elements

I1ID=*i1 id*
Refers to the first level entry for this second level entry. I1ID is required when I2 occurs outside the context of an I1 element.

SEEID=*i1 ids*
Defines a cross reference to one or more other first-level index entries.

SEETEXT=*see also text*
Contains the text of a see also reference. SeeID= and SeeText= can both be specified, if desired.

SORTKEY=*sortkey text*
When specified, the SortKey= text is used to sort the entry, rather than the index entry text itself.

PRIMARY
Indicates that this entry is a primary reference point.

index entry text
The index term itself.

I3 One or more I3 elements.

Usage

Use explicit index entries when the indexing function provided by author-defined classes and phrase markup is not sufficient for your information. Index entries specified in the information content will be used to generate a traditional index. When index entries are specified in an IndexDefinitions element within Prolog or DivProlog, they define index structures that can be used by reference from within the document content. Index entries within IndexDefinition will not appear in a generated index unless specifically referred to.

You can link related index entries together with the SeeID= attribute.

Use the SortKey= attribute to define the character string to be used when sorting the index entries if it is different from the index entry text itself. For example, you may want a number to sort as though it were spelled out.

Index entries are associated with the element that directly contains them, regardless of where in that element they occur. For example, an index entry within a paragraph is associated with the entire paragraph. You can associate an index entry with a specific point within a paragraph by putting it within a phrase element.

You can define your own classes for index entries and assign special meaning and processing to those classes. For example, you could define a class called Search for index entries that are only to be used as search targets in online presentation systems. Classes are defined using the ClassDef element.

You can also use author-defined element classes to create indexes. See “ClassDef (element class definition)” on page 74 for more information.

Examples

```
<p><i2 i1id='relatedconcept'>about</i2>
This paragraph is about a concept.
```

I2 can be contained within the I1 that defines its first-level entry:

```
<i1>concept<i2>about<i2>definition</i1>
```

SGML Definition

```
<!ELEMENT  I2              - O (IndexTerm, I3*)
                                        -- second-level index entry --
>
<!ATTLIST  I2
    I1ID             NAME         #IMPLIED              -- I1 parent --
    Primary          (Primary)    #IMPLIED          -- Primary entry? --
    SeeID            CDATA        #IMPLIED
    SeeAlso          NAMES        #IMPLIED
    SortKey          CDATA        #IMPLIED
    %Common.Atts.NoRefType;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
    RefType          CDATA        #FIXED
                          "I1ID I1 SeeAlso I1
                           &Common.RefType;"
>
```

I3 (third-level index entry)

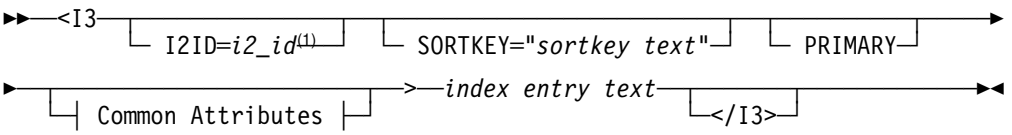
Purpose

The I3 element contains a third-level index entry. An index entry is associated with the element that directly contains it. I3 is valid wherever PhraseLevelElements or ParaLevelElements is valid, or within an IndexDefinitions element within Prolog or DivProlog. I3 outside the context of I2 must use the I2ID= attribute to refer to an I2 element.

Valid Contexts

IndexDefinitions, i2, PhraseLevelElements, ParaLevelElements.

Syntax



Note:
1 I2ID= is required when I3 occurs outside the context of I2.

Attributes and Contained Elements

I2ID=*i2 id*
Refers to the first level entry for this second level entry. I2ID is required when I3 occurs outside the context of an I2 element.

SORTKEY=*sortkey text*
When specified, the SortKey= text is used to sort the entry, rather than the index entry text itself.

PRIMARY
Indicates that this entry is a primary reference point.

index entry text
The indexed term.

Usage

Use explicit index entries when the indexing function provided by author-defined classes and phrase markup is not sufficient for your information. Index entries specified in the information content will be used to generate a traditional index. When index entries are specified in an IndexDefinitions element within Prolog or DivProlog, they define index structures that can be used by reference from within the document content. Index entries within IndexDefinition will not appear in a generated index unless specifically referred to.

Use the SortKey= attribute to define the character string to be used when sorting the index entries if it is different from the index entry text itself. For example, you may want a number to sort as though it were spelled out.

Index entries are associated with the element that directly contains them, regardless of where in that element they occur. For example, an index entry within a paragraph is associated with the entire paragraph. You can associate an index entry with a specific point within a paragraph by putting it within a phrase element.

You can define your own classes for index entries and assign special meaning and processing to those classes. For example, you could define a class called Search for index entries that are only to be used as search targets in online presentation systems. Classes are defined using the ClassDef element.

You can also use author-defined element classes to create indexes. See “ClassDef (element class definition)” on page 74 for more information.

Examples

```
<p><i3 i2id='abouaconcept'>more about</i3>
This paragraph is about a concept.
```

I3 can be contained within the I2 that defines its second-level entry:

```
<i1>concept<i2>about<i3>more<i2>definition</i1>
```

SGML Definition

```
<!ELEMENT  I3              - O (IndexTerm)
                                          -- third-level index entry --
>
<!ATTLIST  I3
    I2ID             NAME         #IMPLIED              -- I2 parent --
    Primary          (Primary)    #IMPLIED          -- Primary entry? --
    SortKey          CDATA        #IMPLIED
    %Common.Atts.NoRefType;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
    RefType          CDATA        #FIXED
                          "I2ID I2
                           &Common.RefType;"
>
```

Kwd (syntactic keyword)

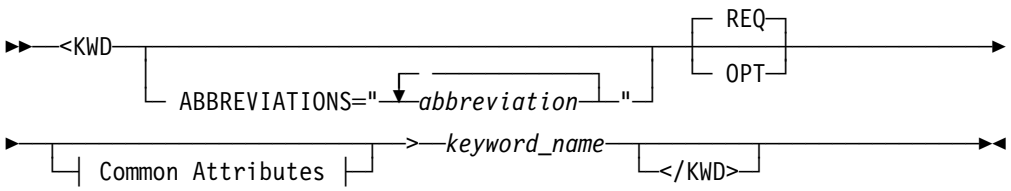
Purpose

Defines a keyword within a syntax definition group, fragment, or syntax phrase.

Valid Contexts

Group, RepSep, SyntaxPhrase.

Syntax



Attributes and Contained Elements

ABBREVIATIONS=*abbreviations*
Lists the valid abbreviations for the keyword.

REQ | **OPT**
Indicates whether or not the keyword is optional. REQ (required) is the default.

keyword_name
The name of the keyword.

Usage

Use Kwd to define keywords within a syntax definition. Keywords are literal values that must be specified exactly as shown in the diagram.

Examples

```
<syntax>Attribute Specification
  <group>Attspec
    <sep>&ssbl;
    <kwd abbreviations="ATT ATTN ANAME">ATTNAME
    <oper>=
    <var>value
  </group>
</syntax>
```

SGML Definition

```
<!ELEMENT  Kwd              - O (%DataPool;)*       -- Syntax keyword -->
<!ATTLIST  Kwd
    Abbreviations    CDATA        #IMPLIED
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;
>
```

LE (language element)

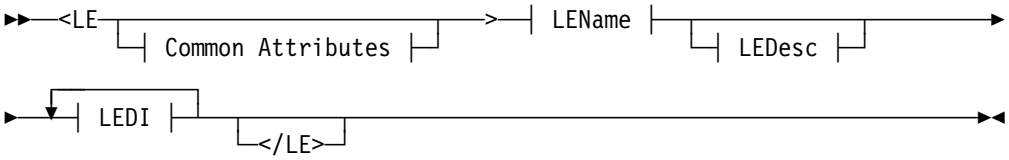
Purpose

LE contains the description of a computer language element, such as a command, within the context of a language element reference section. LE is a specialized version of InfoModule.

Valid Contexts

LERS.

Syntax



Attributes and Contained Elements

LEName
The name of the language element, for example, a command name.

LEDesc
A short description of the language element, intended to be part of the presented title.

LEDI
One or more language element description items.

Usage

Use LE and LERS to create reference information for computer languages such as command sets, programming languages, and the like. The default presentation of language elements is as headed sections, but you can define other presentations in the style definition. For example, you may want to present language elements in a table for quick reference.

Examples

```
<LERS>
<LE id=fubar>
<LEName>FUBAR command
<LEDesc>Slices, dices, makes julien fries
<ledi class=purpose>
<p>FUBAR performs the slicing and dicing function.
<ledi class=format>
<syntax>Fubar command
</syntax>
</LE>
  ⋮
</LERS>
```

SGML Definition

```
<!ELEMENT  LE               - O (LEName, LEDesc?, LEDI+)
                                                  -- Language Element --
>
<!ATTLIST  LE
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
    ApplicationClass NAME         #FIXED    "ComputerLanguage"
>
```

LEDesc (language element description)

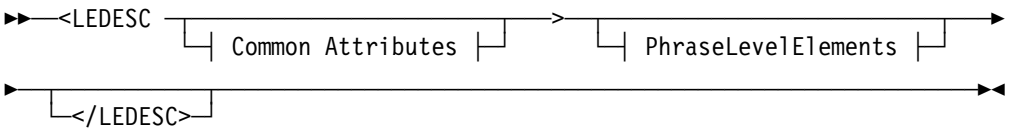
Purpose

Contains a short description of the language element.

Valid Contexts

LE, following LEName.

Syntax



Attributes and Contained Elements

PhraseLevelElements
LEDesc can contain any phrase-level elements.

Usage

Use LEDesc to contain a short description of the language element for possible use in presented titles.

Examples

```
<lers>
<le id=fubar>
<lename>FUBAR command
<ledesc>Slices, dices, makes julien fries
<ledi purpose>
<p>FUBAR performs the slicing and dicing function.
<ledi format>
<syntax>Fubar command
</syntax>
</le>
  ⋮
</lers>
```

SGML Definition

```
<!ELEMENT  LEDesc           - O (%DataPool;)*
                                    -- Language element description --
>
<!ATTLIST  LEDesc
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescription"
>
```

LEDI (language element description item)

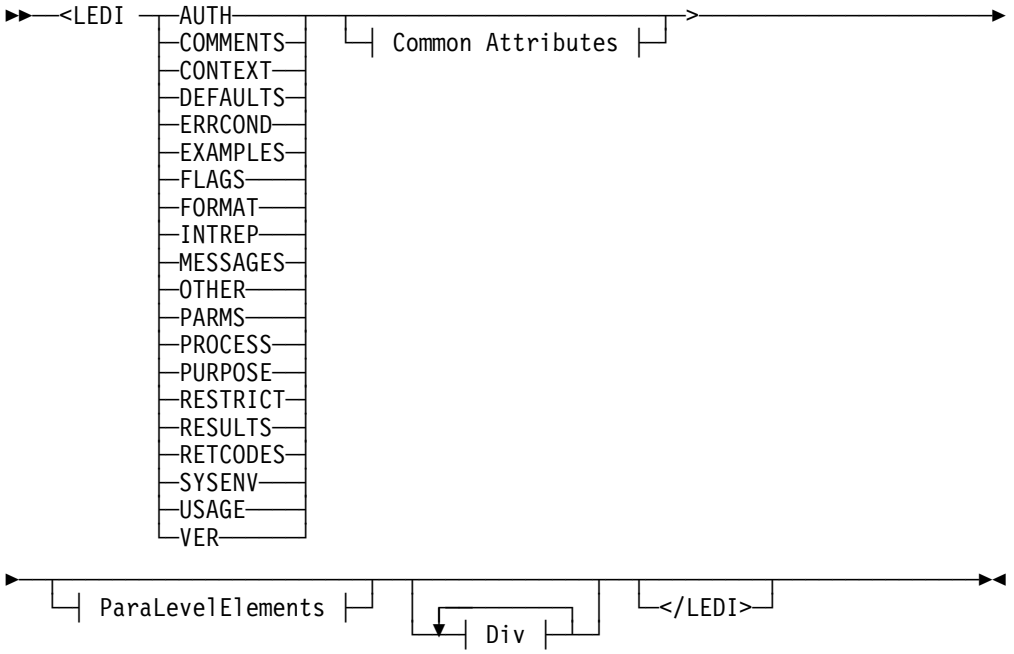
Purpose

Contains a description of one aspect of a language element within a LERS section.

Valid Contexts

LE, following any LEName or LEDesc elements.

Syntax



Attributes and Contained Elements

Class Values

Indicates the class of the LEDI. The class values define what type of information each LEDI contains:

AUTH
Authorization level, e.g., user, superuser, etc.

COMMENTS
Comments about the language element.

CONTEXT
The context in which the language element is used.

DEFAULTS
The defaults for the language element.

ERRCOND
Error conditions.

EXAMPLES
Examples of using the language element.

FLAGS
Control flags.

FORMAT
The format or syntax of the language element.

INTREP
Internal representation, such as control blocks or data structures.

MESSAGES
Any messages related to the language element.

OTHER
An open category.

PARMS
Parameters.

PROCESS
Processing related to the language element.

PURPOSE
The purpose of the language element.

RESTRICT
Restrictions on the use of the language element.

RESULTS
The results of using the language element.

RETCODES
Return codes.

SYSENV
The system environments to which the language element applies.

USAGE
How to use the language element.

VER
Any version information for the language element.

Usage

The LEDIs within a language element description define the structure of the description by subdividing it into subsections with meaningful names.

For each LEDI, the title text is defined separately on a LERSDEF element or in the document style definition. Each LEDI class can have a different generated title or no title. The default presentation style for language element descriptions is as normal divisions, but other styles can be used as well. For example, you can define a table-like presentation for LEs. In that case, the generated LEDI title would be used as the column headings for the columns corresponding to each LEDI class. Also, through the document style definition, you can choose to suppress the presentation of any LEDI class. For example, you may want to only show the format items in a quick reference created from the same source that is used for the full reference document.

LEDIs can contain divisions.

Examples

```
<lers>
<le id=fubar>
<lename>FUBAR command
<ledesc>Slices, dices, makes julien fries
<ledi purpose>
<p>FUBAR performs the slicing and dicing function.
<ledi format>
<syntax>Fubar command
</syntax>
</le>
  ⋮
</lers>
```

SGML Definition

```
<!ELEMENT  LEDI             - O ((%InfoUnitPool;)* | Div+)
                                 -- Language Element Description Item --
>
<!ATTLIST  LEDI
    Class            (Auth |
                      Comments |
                      Context |
                      Defaults |
                      Errcond |
                      Examples |
                      Flags |
                      Format |
                      Intrep |
                      Sep |
                      Messages |
                      Other |
                      Parms |
                      Process |
                      Purpose |
                      Restrict |
                      Results |
                      Retcodes |
                      Sysenv |
                      Usage |
                      Ver )
                                     #REQUIRED
    %Common.Atts.NoClass;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "LanguageElementItem"
>
```

Legend

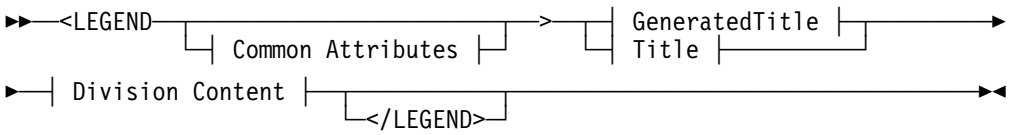
Purpose

Legend is intended to contain an explanation of any special notations used in the document, such as within graphics, tables, figures, or other specialized information.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the Legend division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

Division Content

Legend has the same content rules as Div elements.

Usage

Examples

```
<frontmatter>
<legend><generatedtitle>
<p>The following symbols have special meaning in this document:
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  Legend           - O (%SpecialDiv;) >
<!ATTLIST  Legend
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

LEName (language element name)

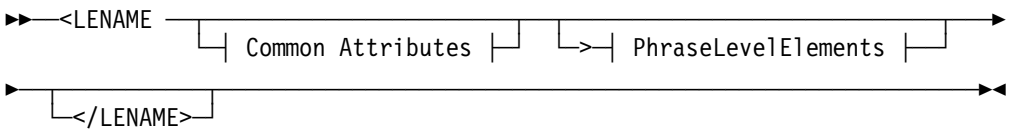
Purpose

Contains the name of a language element.

Valid Contexts

LE.

Syntax



Attributes and Contained Elements

PhraseLevelElements

LEName can contain any phrase-level elements.

Usage

Use LENAME for the name of the language element, for example, the command name.

Examples

```
<lers>
<le id=fubar>
<lename>FUBAR command
<ledesc>Slices, dices, makes julien fries
<ledi purpose>
<p>FUBAR performs the slicing and dicing function.
<ledi format>
<syntax>Fubar command
</syntax>
</le>
  ⋮
</lers>
```

SGML Definition

```
<!ELEMENT  LEName           - O (%DataPool;)*
                                          -- Language Element Name --
>
<!ATTLIST  LEName
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleName"
>
```

LERS
(language element reference section)

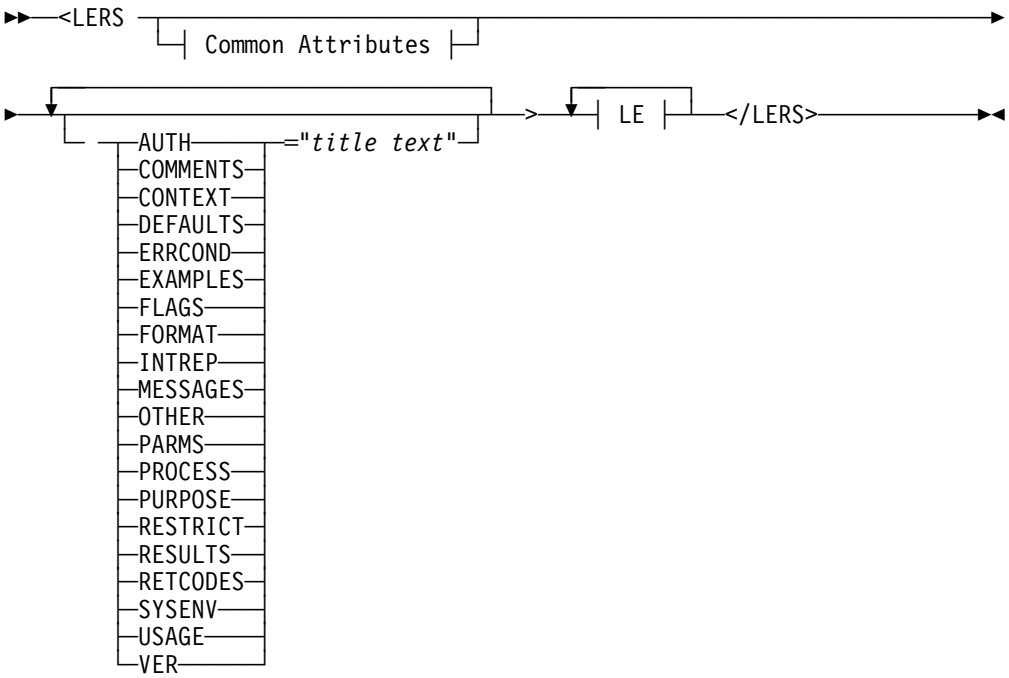
Purpose

Contains a set of computer language element descriptions intended to form a language reference section or document.

Valid Contexts

ParaLevelELements.

Syntax



Attributes and Contained Elements

classname=title text
: For each LEDI class, you can define the generated title for the LEDI elements. See “LEDI (language element description item)” on page 166 for details on the individual parameters.

LE One or more language element (LE) elements.

Usage

Use LERS to contain reference information for computer languages such as programming languages, command sets, and the like. Use the more generic Modular Information elements for other sorts of modular information. A LERS section contains one or more language element descriptions (LE). LEs can also be organized within object libraries and used by reference from within a LERS section using the common ContentRef= attribute.

The default presentation style for LERS is as normal divisions, but you can define other presentation styles for LERS. For example, you may want to present each language element as a row in a table for quick reference.

Examples

```
<lers>
<le id=fubar>
<lename>FUBAR command
<ledesc>Slices, dices, makes julien fries
<ledi purpose>
<p>FUBAR performs the slicing and dicing function.
<ledi format>
<syntax>Fubar command
</syntax>
</le>
  ⋮
</lers>
```

SGML Definition

```
<!ELEMENT  LERS             - - (LE+)
                                -- Language Element Reference Section --
>
<!ATTLIST  LERS
    %lersdef.atts;
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleList"
    ApplicationClass NAME         #FIXED    "ComputerLanguage"
>
```

LERSDef (LERS property definition)

Purpose

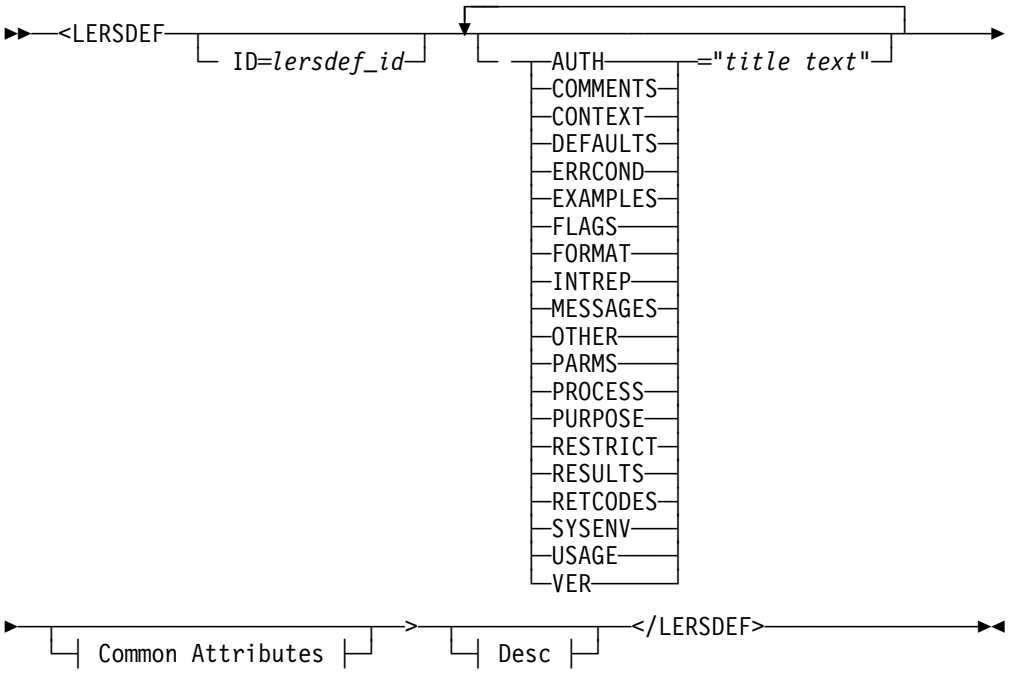
Defines property values for a language element reference section (LERS).

Valid Contexts

PropertyDefinitions.

Syntax

LERSDef (LERS property definition)



Attributes and Contained Elements

ID=*lersdef_id*

Property definitions without an ID= define the default properties for the division or document within which they occur.

classname=**"***title text***"**

For any LEDI class, defines the generated title text to be used for that class of LEDI. You can define an empty title.

For any LEDI class not specified, the default title text defined in the active style will be used. See “LEDI (language element description item)” on page 166 for a complete description of the LEDI classes.

Desc

An optional text description of the LERS definition. This is intended to provide documentation about why a certain set of properties have been defined.

Usage

Use LERSDef to define the titles to be generated for different classes of LEDI elements, and to define values for common properties.

Examples

```
<lersdef id=taglers
         format="Syntax"
         parms="Attributes and Contained Elements"
         usage="Usage"
         other="SGML Definition"
         context="Related Elements">
```

SGML Definition

```
<!ENTITY % lersdef.atts
   "Auth             CDATA        #IMPLIED
    Comments         CDATA        #IMPLIED
    Context          CDATA        #IMPLIED
    Defaults         CDATA        #IMPLIED
    Errcond          CDATA        #IMPLIED
    Examples         CDATA        #IMPLIED
    Flags            CDATA        #IMPLIED
    Format           CDATA        #IMPLIED
    Intrep           CDATA        #IMPLIED
    Messages         CDATA        #IMPLIED
    Other            CDATA        #IMPLIED
    Parms            CDATA        #IMPLIED
    Process          CDATA        #IMPLIED
    Purpose          CDATA        #IMPLIED
    Restrict         CDATA        #IMPLIED
    Results          CDATA        #IMPLIED
    Retcodes         CDATA        #IMPLIED
    Sysenv           CDATA        #IMPLIED
    Usage            CDATA        #IMPLIED
    Ver              CDATA        #IMPLIED"
>
<!ELEMENT  LERSDef          - O (Desc?)
                       -- Language Element Reference Section Definition --
>
<!ATTLIST  LERSDef
    %lersdef.atts;
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "PropertyDefinition"
>
```

LI (list item)

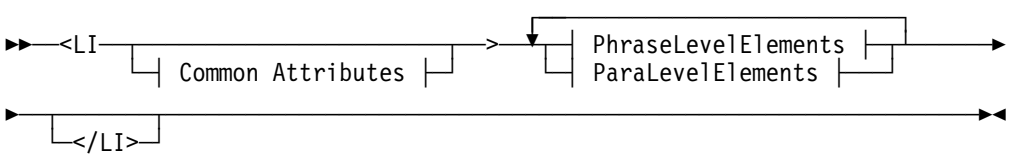
Purpose

Contains a single list item within an ordered list, unordered list, or note list.

Valid Contexts

OL, UL, NoteList, StepNotes, LIBlock.

Syntax



Attributes and Contained Elements

PhraseLevelElements
ParaLevelElements
List items can contain any paragraph- or phrase-level elements.

Usage

Use LI within lists. List items can be organized within a list using a list item block element to group list items together.

Examples

```
<ol>
<li>bread
<li>milk
<li>peanut butter
</ol>
```

SGML Definition

```
<!ELEMENT  LI              - O (%InfoPool;)*          -- List Item -->
<!ATTLIST  LI
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ListItem"
>
```

LIBlock (list item block)

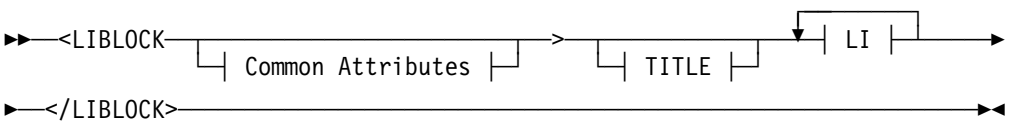
Purpose

Organizes list items within an ordered or unordered list.

Valid Contexts

UL, OL, NoteList, StepNotes.

Syntax



Attributes and Contained Elements

Title
The title of the block of list items. When a title is specified, the Title start tag must be specified.

Usage

Use LIBlock to organize list items within a list, or to define blocks of list items to be connected with a Bridge element.

Migration Note

The use of LIBlock and Bridge replaces the List Part (LP) element from DCF BookMaster.

Examples

```
<ul>
<liblock><title>First of two blocks
<li>In first block
<li>In first block
</liblock>
<bridge>
<p>This bridge connects the first block
with the second block.
</bridge>
<liblock>
<li>In second block
<li>In second block
</liblock>
</ul>
```

SGML Definition

```
<!ELEMENT  LIBlock          - O (Title?, (LI | Bridge)+)
>
<!ATTLIST  LIBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ListItemBlock"
>
```

LibRecord (document library definition)

Purpose

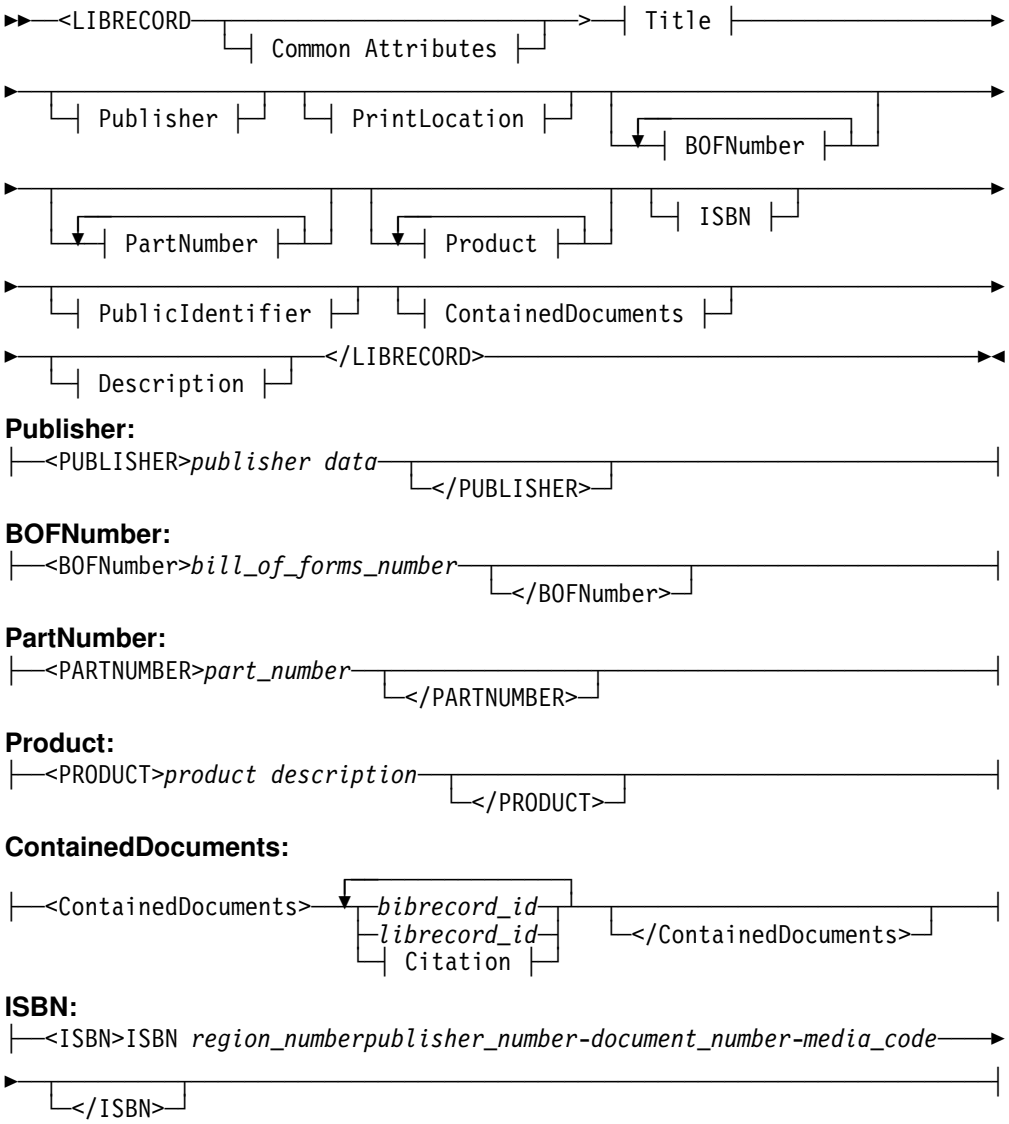
Library records structure and organize the information about libraries or collections of documents.

Use IBMLibRecord to define a collection of IBM documents.

Valid Contexts

BibList, BibRecords, Citation, BibRecord, IBMBibRecord, IBMLibRecord.

Syntax



Attributes and Contained Elements

Title
The title of the library.

Publisher
Contains the name of the publisher followed by an optional address element.

BOFNumber
The bill of forms (BOF) number assigned to this library.

PartNumber
The part number assigned to this library.

ISBN
The ISBN number assigned to this library.

PublicIdentifier
The SGML public identifier assigned to this library. This is the same public identifier used in entity declarations for the system object that represents this library.

ContainedDocuments
Defines the documents contained in this library and the default order for presenting the documents when the library definition is presented.

The contained documents can be identified directly using Citation elements, or indirectly by specifying BibRecord or LibRecord IDs.

The order the contained documents are specified in ContainedDocuments defines the default order for presenting the library contents.

Description
A Desc element containing a description of the library.

Usage

Use LibRecord to define the bibliographic information about a library or other collection of documents. You can use the common Class= attribute and the ClassDef element to define different classes of LibRecord to correspond to different classes of collection. For example, you may have product libraries that are themselves collected into larger libraries of libraries. You could define a class of 'CollectionKit' for libraries of libraries and then use LibRecord elements to define the contents of a given collection kit.

LibRecords can also be specified in a BibRecords section or object container and used by reference from within a document, for example, from Citation elements.

When Linkends= is used with the anchor role DescribedDocument, the LibRecord element becomes the anchor of a link to the library described. When the unified name space option is in effect, the Linkends= value can be the name of the entity that represents the described library. When the unified name space option is not in effect, you must create Nameloc element to map the ID reference used in the Linkends= value to the entity name (see “Nameloc (named location)” on page 220 for details on using the Nameloc element).

The default hyperlink style is for the LibRecord and all references to it to be active hyperlinks to the library.

When LibRecords are specified within Citation, the LibRecords are collected for use in a generated bibliography (see “BibList (bibliography entry list)” on page 56).

It is intended that the public identifier of the library entity be used by the presentation system to locate the actual system object, but specific presentation systems may define application-specific data to be specified as the system identifier of the library entity if they do not support the use of public identifiers. The public identifier can be included in the LibRecord itself as a way of keeping a libraries's formal public identifier definition with the rest of its bibliographic information. This could allow, for example, the automatic generation of entity declarations for libraries described by LibRecord elements.

See Appendix A, “Proposed IBM Standard for Formal Public Identifiers” on page 332 for a description of the formal public identifier standard defined by the IBM Corporation for its internal and external use.

Examples

```
<librecord id=sgmlbookielib>
 <Title>
 <shorttitle>IBM ID Doc Library</shorttitle>
Publishing Solutions
IBM ID Doc Library
 <Publisher>TIC Corporation
  <address>
  </address>
 </publisher>
 <PrintLocation>USA
 <BOFNumber>SBOF-6000-00
 <PublicIdentifier>+//ISBN 0-19-9999
 //LIB SBOF-6000-00
 /IBM ID Doc Library
 /drmacro@ralvm13.vnet.ibm.com
 /Publishing Solutions
 //EN
 <ContainedDocuments>bkmbookieug sbkmbookieref
 </ContainedDocuments>
 <Desc>Documentation for the IBM ID Doc language.
 IBM ID Doc is an SGML language for creating printed and online
 technical information.
</librecord>
```

SGML Definition

```
<!ELEMENT  LibRecord        - -
                      (Title, Publisher?, PrintLocation?,
                       BOFNumber?, PartNumber?,
                       Product*, ISBN?, PublicIdentifier?,
                       ContainedDocuments?, Desc?)
                                                -- Library Record --
>
<!ATTLIST  LibRecord
    %LibRecord.Atts;
>
<!ELEMENT  IBMLibRecord     - -
                      (Title, Publisher?, PrintLocation?,
                       IBMBOFNumber?, IBMPartNumber?,
                       Product*, ISBN?, PublicIdentifier?,
                       ContainedDocuments?, Desc?)
>
<!ATTLIST  IBMLibRecord
    %LibRecord.Atts;
>
<!ELEMENT  ContainedDocuments
                               - O (#PCDATA | Citation )*
>
<!ATTLIST  ContainedDocuments
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
```

```
                        "#CONTENT (name | #GI) &Common.LexModel;"
    RefType          CDATA        #FIXED
                        "#CONTENT (&BibRecord; | &LibRecord;)
                         &Common.RefType;"
            -- The order BibRecord IDs are specified defines the
               default order for presenting the list of documents
               in a generated bibliography --
>
<!ELEMENT  BOFNumber        - O (%DataPool;)* -- Bill of Forms number -->
<!ATTLIST  BOFNumber
    %Common.Atts;
>
<!ELEMENT  IBMBOFNumber     - O (%DataPool;)*
                                        -- IBM Bill of Forms number --
>
<!ATTLIST  IBMBOFNumber
    %Common.Atts;
>
<!ELEMENT  Product          - O (%DataPool;)* >
<!ATTLIST  Product
    %Common.Atts;
>
```

Lines (text with significant line boundaries)

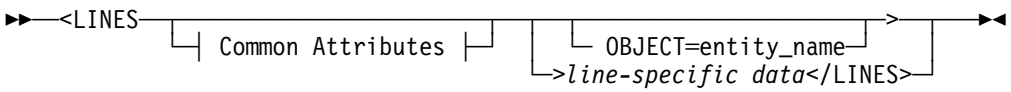
Purpose

Contains text for which the input line (record) boundaries are significant and must be preserved or indicated when presented. Lines is a special case of ObjectContainer.

Valid Contexts

ParaLevelElements

Syntax



Attributes and Contained Elements

OBJECT=*entity_name*

The name of the external data entity that contains the line-specific data. When Object= is specified, it is an error to specify any data or the Lines end tag.

line-specific data

The data for which line boundaries are to be preserved. All significant record ends will be preserved in the presentation. How significant record ends are presented is a matter of style, the default style being to present each line separately on its own output line.

Record boundaries can be specified explicitly by using the character reference for record end and record start like so: `&#RE;&#RS;`.

On record-oriented systems, two input lines can be concatenated using a reference to the function character '&#CONT' without the trailing semicolon. In this specific case, SGML treats the record end as the end of the character reference and not as a signficant record end, effectively concatenating the line following it to the end of the previous line.

Note: The #CONT function character is defined by IBM ID Doc in its SGML declaration; it is not a standard SGML function character.

Usage

Use Lines to contain information where the line boundaries are important and must be preserved in the presentation. A typical use of Lines is to present poetry.

Examples

```
<figure class=poetry>
<title>The Song of Wandering Aengus
<lines>
I went out to the hazel wood
Because a fire was in my head
I cut and peeled a hazel wand
And hooked a berry to a thread.
And when white moths were on the
Wing, and moth-white stars were flickering out,
I dropped the berry into a stream
And caught a little silver trout.
&rbl;
  ⋮
</lines>
</figure>
```

SGML Definition

```
<!ELEMENT  Lines            - - (%DataPool;)*   -- Line-specific data -->
<!ATTLIST  Lines
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    Notation         (LineSpecific)
                                  #FIXED    "LineSpecific"
    ApplicationClass NAME         #FIXED    "Lines"
>
```

Link (explicit hyperlink)

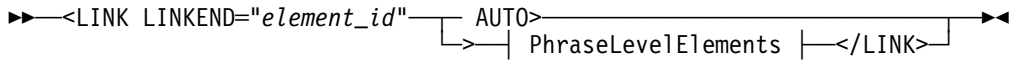
Purpose

Link lets you make any phrase a hyperlink to any place in the document, another document, or some non-text object, such as a multimedia presentation.

Valid Contexts

MultiMediaObject and wherever phrase-level elements are valid.

Syntax



Attributes and Contained Elements

LINKEND=*element_id*

The ID= value of the element being linked to or a HyTime location element that ultimately locates the object or objects being linked to.

AUTO

When **AUTO** is specified, the link behaves like a cross reference and the link text is generated automatically. When AUTO is specified, it is an error to specify the Link end tag.

Usage

Link can point to either another element in the same document, or it can point to a LinkDesc element, through which it can link to almost anything. What you can link to via LinkDesc is limited only by the online presentation system you use.

Note: IBM ID Doc neither defines nor limits the types of things you can link to from a document. The linking you can do is determined by the online presentation system you are using.

As a rule, it is worth using LinkDesc when something will be linked to more than twice within the same document, because the indirection provided by LinkDesc makes maintaining those links easier. See “LinkDesc (linked location description)” on page 184 for more information.

Examples

Linking directly to another element:

```
<p>The <link linkend=aboutlink>LINK element</link> lets you....
  ⋮
<p id=aboutlink>LINK is a generic, unrestricted hyperlink....
```

Linking to another element using LinkDesc:

```
<p>The <link linkend=LdescAboutLink>LINK element</link> lets you....
  ⋮
<linkdesc id=LdescAboutLink>A link to the description of LINK
  <nmlist>aboutlink
</linkdesc>
  ⋮
<p id=aboutlink>LINK is a generic, unrestricted hyperlink....
```

Linking to an element in another document:

```
<!-- In source document -->
<!--   Entity Declarations:  -->
<!ENTITY BookMasterRef PUBLIC "-//IBM//DOCUMENT PUB SC31-6001
                                      /BookMaster Reference//EN" >
  ⋮
<nameloc id=aboutlink-local>
<nmlist docname=BookMasterRef>aboutlink</link>
</nameloc>
<p>The <link linkend=AboutLink-local>LINK element</link>
lets you....
  ⋮
<!-- In document BookMasterRef -->
<p id=aboutlink>LINK is a generic, unrestricted hyperlink....
```

SGML Definition

```
<!ELEMENT  Link             - - (%DataPool;)*
                                    -- Explicit hyperlink phrase --
>
<!ATTLIST  Link
    LinkEnd          NAME         #REQUIRED
    LinkPhrase       (Auto)       #CONREF
    ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    XRefText         CDATA        #IMPLIED
    InfoType         CDATA        #IMPLIED
    %Hyperlink.Attributes;
    %Property.Attributes;

    InfoMaster       NAME         #FIXED    "HyperLink"
    -- HyTime attributes: --
    HyTime           NAME         #FIXED    "ilink"
    HyNames          CDATA        #FIXED
                          "linkend LinkEnd &Common.HyNames;"
    RefType          CDATA        #FIXED
                         "#ALL #ANY &Common.RefType;"
                       -- No REFLEVEL restrictions (see 10744 6.5.3) --
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    refrange         CDATA        #FIXED    "#ALL I"   -- indirection --
>
```

LinkDesc (linked location description)

Purpose

LinkDesc contains HyTime location elements used to provide indirect methods of linking to hyperlink anchors and associates a description of the anchors or the reason for the indirection with the location elements. The contained location elements can serve several purposes. They can provide an indirect method of refering to another SGML element in order to protect hyperlinks from changes in the target elements, they can associate a single ID with multiple elements, and they can associate an ID with a query specification.

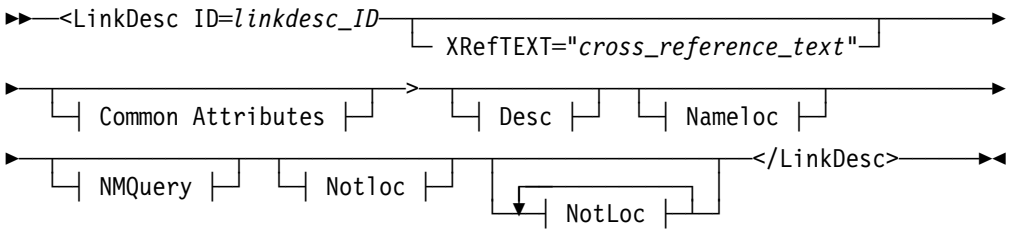
If the necessary HyTime support is provided, the contained location elements can also be the start more complex HyTime location ladders.

Only support for the base location module of HyTime is required, but specific implementations may add support for other HyTime location elements by adding them to the content of LinkDesc.

Valid Contexts

LinkDescriptions.

Syntax



Attributes and Contained Elements

ID=*Ldesc_ID*
The ID of this link description.

XRefTEXT=*cross_reference_text*
Defines the text to be used for cross references to the elements linked to from this LinkDesc.

Desc
Contains a description of the link. This information may be used by an online presentation system to provide information about a link. The description is optional.

Nameloc
Nameloc contains one or more NMList or NMQuery elements.

NMQuery
Contains a query statement. The specifics of the query statement are a function of the online presentation or processing system. See “NMQuery (named query)” on page 225 for details.

You can specify multiple NMQuery elements using different notations for the same query in order to support different application-specific query notations.

Notloc
Contains a definition of the location of a link anchor in some application-specific notation.

Usage

Location elements let you define links that are more complex or more flexible than a simple cross reference or direct link. Use location elements under the following circumstances:

- When an element is a target of many links and you want to avoid having to re-work every link if the target element changes.

- When you want to link to multiple places from a single point in your document, especially from a cross-reference. When a cross-reference is made via a location that points to more than one element or that uses a NMQuery, you should use the common XRefText= attribute to define the cross reference text so that the processor has something meaningful to use for the generated text of the cross reference.
- When you want a link to be a pre-defined associative or query link. Using associative links in this way lets you create precise links that require little maintenance.
- When you want to create a link outside your information, such as to a separate program.

Examples

Link to multiple elements in the same document:

```
<p>The <link linkends=LdescAboutLink>LINK element</link> lets you....
  ⋮
<linkdesc>
A link to all the
places discussing links
  <nameloc id=LdescAboutLink>
  <nmlist>aboutlink aboutxref
  </nameloc>
</linkdesc>
  ⋮
<p id=aboutlink>LINK is a generic, unrestricted hyperlink....
  ⋮
<hd id=AboutXref>XRef Explained
```

SGML Definition

```
<!ELEMENT  LinkDesc         - - (Desc?, (nameloc | nmquery)?, NotLoc*)
                                                  -- Link Description --
>
<!ATTLIST  LinkDesc
    %Common.Atts.NoHyTime;

    InfoMaster       NAME         #FIXED    "LinkDescription"

    -- HyTime attributes: --
    HyTime           NAME         #FIXED    nameloc
    ordering         (ordered)    ordered
              -- Up to presentation system to define meaning of order --
    set              (set)        set                 -- Ignore dupes --
    aggloc           (nagg)       nagg               -- Not aggregate --
    spanloc          (nspan)      nspan                   -- No spans --
    limsort          (nlimsort)   nlimsort        -- No limit sorting --
>
```

LinkEnds, AnchorRoles, and HyTime Attributes (hyperlink target reference attributes)

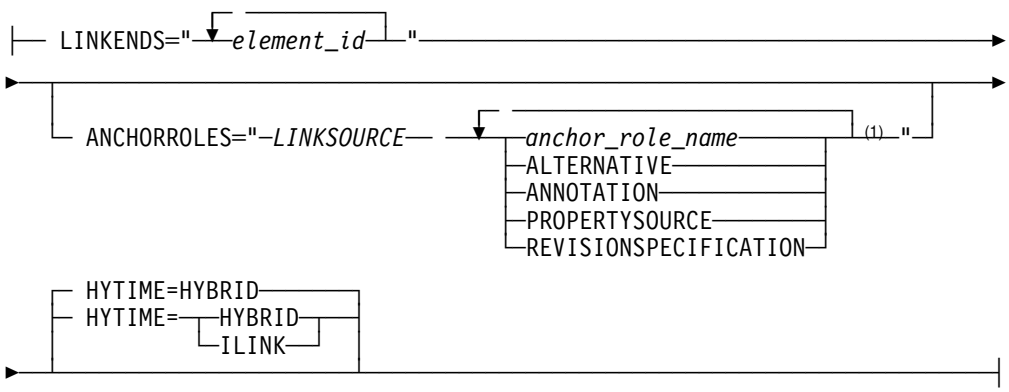
Purpose

Establishes hyperlinks for various purposes from any element to any supported hyperlink target, including other elements, other documents, and non-SGML data entities.

Linkends= conforms to the HyTime linkends attribute form. AnchorRoles= conforms to the HyTime anchrole attribute form. When Linkends= and AnchorRoles= are specified, the element on which they are specified conforms to the HyTime ilink element form.

Syntax

LinkEnds=:



Note:
[1] Each anchor role name can only be used once

Attributes and Contained Elements

Linkends=*element IDs*

Lists the IDs of the elements that satisfy the corresponding anchor roles listed in AnchorRoles=. There must be one and only one ID per anchor role name, except that you can omit the ID of the LINKSOURCE role when the link source is the current element (which is normally the case).

AnchorRoles=*anchor role names*

Takes the names of anchor roles, either IBM ID Doc defined or author defined. The anchor role name 'LinkSource' always refers to the current element. It must be the first anchor role name unless the current element's ID is listed in the Linkends= value.

anchor_role_name

The name of an author-defined anchor role name, defined using the AnchorRoleDef element.

ALTERNATIVE
ANNOTATION
PROPERTYSOURCE
REVISIONSPECIFICATION
IBM ID Doc-defined anchor roles, used to establish IBM ID Doc-defined hyperlink relationships.

HYTIME=HYBRID | ILINK
The HyTime= attribute indicates what HyTime architectural form a given element conforms to. Under normal circumstances, most IBM ID Doc elements conform to the HyTime HyBrid form, which simply means that they can contain HyTime elements, but that they have no special HyTime meaning themselves.

However, when the Linkends= and AnchorRoles= attributes are specified, the element becomes a HyTime independent link (ilink) element, and must be identified as such to HyTime, which you do by specifying the value `'ilink'` for the HyTime= attribute.

Usage

LinkEnds= is used to establish hyperlinks between an element (the link source, in IBM ID Doc terms), and other objects. The hyperlinks represent specific relationships, defined either by IBM ID Doc or by you as author of the document. The purpose of each hyperlink is indicated by a matching anchor role name specified on the AnchorRoles= attribute.

IBM ID Doc defines four specific hyperlink relationships:

- Annotations
- Retrieval alternatives
- Property sources
- Revision specifications

The annotations relationship links an element with other elements that serve to annotate the source element. For example, you might link a paragraph to a generic safety notice by creating an annotation hyperlink from the paragraph to the Safety element, thus indicating that the safety notice is an annotation of the paragraph. See “Annotation Anchor Role (annotation link)” on page 48 for more information on annotation links.

The retrieval alternatives relationship lets you link alternate versions of the same element together. For example, in a multi-language document, you might link the various different language versions of a paragraph together using a retrieval alternatives link, thus making it very clear that these different elements represent a single logical object. See “Alternate Anchor Role (retrieval alternative link)” on page 286 for more information on retrieval alternative links.

The property source relationship links an element to another element from which it will inherit properties. A property source hyperlink is normally used to refer from an element to a PropertyDef element, which has been used to define a set of properties for re-use. For example, you might need to specify a complex Boolean expression for a Version= attribute value, and rather than repeat that value on every element that will need it, you put it on a PropertyDef element and then refer to that PropertyDef via a property source link. See “PropertySource Anchor Role

(property reference link)" on page 275 for more information on property source links.

The revision specification links an element to one or more Revision elements that define the details of a given revision. When a revision specification link is used, the Status= property attribute must also be specified. See "RevisionSpecification Anchor Role (revision specification link)" on page 289 for more information on revision specification links.

When specifying the AnchorRoles= values, the anchor role names must match the Linkends= values one-to-one, with the exception of the LinkSource role. Unless you explicitly list the source element's own ID in the Linkends= value, you must specify the anchor role name LinkSource as the first role name in the AnchorRoles= value. This indicates that the element on which Linkends= is specified satifies that anchor role.

Note that to create a hyperlink relationship to multiple targets requires the use of a Nameloc element to map the single ID name in Linkends= to multiple IDs. For example, to create a retrieval alternative hyperlink to more than one alternate, you must use a Nameloc to name the list of actual alternate elements.

Note To Authors

It is expected that IBM ID Doc authoring support will include editor functionality to manage the details of specifying the different hyperlinks and coordinating the Linkends= values with the AnchorRoles= values, as well as managing any required Nameloc elements needed to create links to multiple targets for a given anchor role.

Examples

The following example shows a simple property source link:

```
<p linkends   ="propref-ver1r2" HyTime=ilink
   anchorroles="linksource propertysource">
```

The following example shows a retrieval alternative link to multiple alternates:

```
<div>Sample Division
<divprolog>
<objectLibrary>
 <p id=para-alt-1>Alternate one
 </p>
 <p id=para-alt-2>Alternate two
 </p>
</objectlibrary>
</divprolog>
  ⋮
<nameloc id=para-alts>
 <nmlist>para-alt-1 para-alt-2
</nameloc>
<p linkends   ="para-alts" HyTime=ilink
   anchorroles="linksource alternate">Alternate zero
```

The following example shows the use of an author-defined anchor role.

```
<div>Sample Division
<divprolog>
<propertydefinitions>
 <anchorroledef rolename=widgetparent>
  <semantic>Use the WidgetParent role to link a widget description
  to the decscription of its parent widget.
  </semantic>
 </anchorroledef>
</propertydefinitions>
</divprolog>
  ⋮
<div linkends   ="maindialog-widget" HyTime=ilink
     anchorroles="linksource widgetparent">
 <title>Dialog Field Widget
  ⋮
</div>
```

SGML Definition

```
    LinkEnds         NAME         #IMPLIED
```

Litdata (literal data)

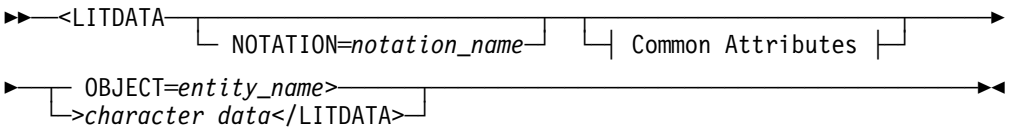
Purpose

Contains or refers to literal data in a specific notation. No SGML markup within LITDATA is recognized except for the the end tag open delimiter (</).

Valid Contexts

ParaLevelElements

Syntax



Attributes and Contained Elements

NOTATION=*notation_name*

The name of the notation that the data is in. The notations supported are defined by the specific implementation of IBM ID Doc you are using, but typical notations include:

Cprog

C and C++ program source.

ASCII
EBCDIC

Indicates data using a specific ASCII or EBCDIC code page. The specific code page is defined using notation attributes on the entity declaration

itself.

The processing application uses the notation name to determine what special processing is needed to present the literal data. For example, in the case of C program code, the square bracket and curly bracket characters may need special treatment.

OBJECT=*entity_name*
Specifies the name of an external data entity that contains the literal data. When OBJECT= is specified, the Litdata end tag cannot be specified.

Usage

Use Litdata to contain or refer to data that is not to be parsed by the SGML parser and that may need special processing in order to properly present the data, such as character translation or use of special code pages.

When the OBJECT= attribute is used to refer to an external entity, the Litdata end tag cannot be specified. Do not specify a notation when refering to an external entity because the entity will have a notation defined on its entity declaration.

The typical use of Litdata is to contain or refer to samples of programming code.

When literal data is included inline in the document, the end tag open open delimiter ('</') will end the Litdata element, regardless of the context of those characters.

Litdata is a special case of the ObjectContainer element.

Examples

```
<figure class=codesamp>
<caption>C language program fragment
<Litdata name=cprog1>
</figure>
```

SGML Definition

```
<!ENTITY % LitData.Notations
    "%ComputerLanguage.Notations; | %TextLanguage.Notations; |
     LineSpecific"
>

<!ELEMENT  Litdata          - O CDATA                -- Literal data -->
<!ATTLIST  Litdata
    Object           ENTITY       #CONREF
    Notation         (%LitData.Notations;)
                                  #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    ApplicationClass NAME         #FIXED    "LiteralData"
>
```

LongQuote (quotation that stands alone)

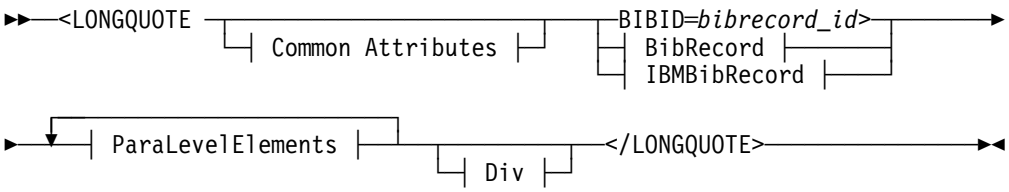
Purpose

Contains an excerpt from another document that is more than a phrase.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

BIBID=*bibrecord_id*
The ID of the BibRecord element that defines the source of the quotation. You must either specify BibID= or include a BibRecord element within the LongQuote element.

BibRecord
IBMBibRecord
Contains the bibliographic information for the source of the quotation.

Usage

Use the LongQuote element to contain material excerpted from another source that can stand alone without a defining context. This is usually the case when the quotation is of considerable length. A LongQuote may be of considerable length and can contain almost any element, including divisions.

In addition to the excerpted material, the LongQuote element must either refer to a BibRecord element or contain the BibRecord element that identifies the source of the excerpt.

Examples

```
⋮
<p>John Kennedy's words of thirty years ago still inspire the listener:
<longquote bibid=jfk0161>
<p>We observe today not a victory of party but a celebration of freedom
— symbolizing an end as well as a beginning — signifying
renewal as well as change.
For I have sworn before you and Almighty God the same solemn oath our
forebears prescribed nearly a century and three-quarters ago.
⋮
<p>Let every nation know, whether it wishes well or ill, that we shall
pay any price, bear any burden, meet any hardship, support any friend,
```

```
oppose any foe to assure the survival and success of liberty.
</longquote>
<p proc=display>To see a video of this speech,
<link linkends="jfkvideo0161">click here</link>.
```

SGML Definition

```
<!ELEMENT  LongQuote        - O ((%InfoUnitPool;)*, Div*) >
<!ATTLIST  LongQuote
    BibID            NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Excerpt"
>
```

ManPage (Unix manual page)

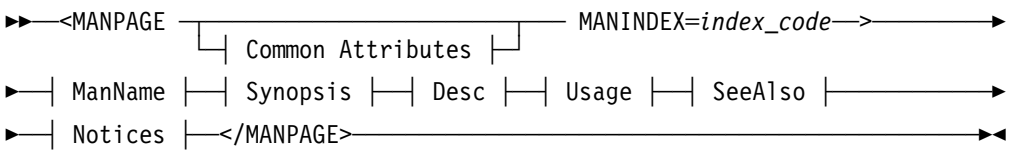
Purpose

ManPage contains a single Unix manual page. ManPage is a specialized version of InfoModule. ManPage represents the man page structure defined by the Davenport group.

Valid Contexts

ManPageCollection.

Syntax



Attributes and Contained Elements

MANINDEX=*index_code*
The man index code for the manual page.

ManName
The name of the object being described in the man page.

Synopsis
Describes the syntax of the command.

Desc
Contains the description of the command.

Usage
Explains how to use the object

SeeAlso
Contains references to other information.

Notices
Contains notices such as restrictions and authors.

Usage

Use ManPage to create Unix manual pages that will not be used in any other context or that must conform to either the OSF document types or to other document types that conform to the Davenport specifications. ManPage is a specialized version of InfoModule. LERS language elements and InfoModules can also be used to create man pages and provide more flexibility than ManPage. The purpose of ManPage is to define and enforce the content rules for man pages defined by the Committee for Common Man subgroup of the Davenport group.

Note to Readers

This definition is a placeholder pending development of an actual man page DTD by the Davenport committee.

Examples

```
<ManPage manindx="1(a)">
<manname>snmpd Daemon
<synopsis>
<p>Provides simple network management protocol services.
</ManPage>
```

SGML Definition

```
<!ENTITY % CommonMan-ManPage
    "ManName, ManSynopsis, ManDesc, ManUsage, ManSeeAlso, ManNotices"
                                          -- temporary place holder --
>

<!ELEMENT  ManPage          - - (%CommonMan-ManPage;) >
<!ATTLIST  ManPage
    ManIndex         CDATA        #REQUIRED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
>

<!ELEMENT  ManName          - O (%DataPool;)* >
<!ATTLIST  ManName
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANName"
>

<!ELEMENT  ManSynopsis      - O (%InfoPool;)* >
<!ATTLIST  ManSynopsis
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANSynopsis"
>
```

```
<!ELEMENT  ManDesc          - O (%InfoPool;)* >
<!ATTLIST  ManDesc
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANDesc"
>

<!ELEMENT  ManUsage         - O (%InfoPool;)* >
<!ATTLIST  ManUsage
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANUsage"
>

<!ELEMENT  ManSeeAlso       - O (%DataPool;)* >
<!ATTLIST  ManSeeAlso
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANSeeAlso"
>

<!ELEMENT  ManNotices       - O (%InfoUnitPool;)* >
<!ATTLIST  ManNotices
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANNotices"
>
```

Mark (marked note definition)

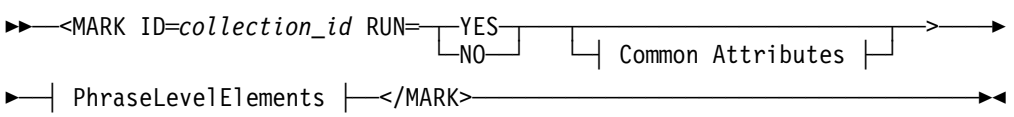
Purpose

Defines a named collection of marked notes within a PropertyDefinition or RevisionInfo section. You must define at least one collection in a document in order to use marked notes. All marked notes must be associated with a collection.

Valid Contexts

RevisionHistory.

Syntax



Attributes and Contained Elements

ID=*collection_ID*
Defines the mark collection ID.

RUN=YES | NO
Determines whether or not the marked notes in this collection should be processed. When **RUN=NO**, the notes in this collection will be ignored.

PhraseLevelElements
Contains a description of the collection.

Usage

Use Mark to define a collection of marked notes. You can define different collections of notes to correspond to different releases of a document or to different types of things that may use the same classes and actions. For example, you if you are using marked notes to track changes and generate a summary of changes, you might define different collections for different releases or drafts of a document. Or if you are tracking changes from different sources, you might define a collection for each different source.

See *IBM ID Doc User's Guide* for more information on using marked notes.

Examples

```
<RevisionHistory>
<mark id=v1r2changes run=yes>
This mark is used to track changes for Version 1 Release 2.
</RevisionHistory>
```

SGML Definition

```
<!ELEMENT  Mark             - O (%DataPool;)* >
<!ATTLIST  Mark
    ID               NAME         #REQUIRED
    run              (yes|no)     #REQUIRED
    %Common.Atts.NoID;
>
```

MarkAction (marked note action definition)

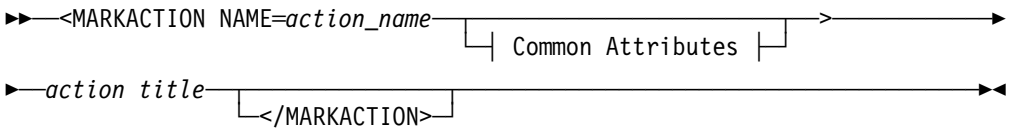
Purpose

Defines a marked note action within MarkDesc.

Valid Contexts

MarkDesc.

Syntax



Attributes and Contained Elements

NAME=*class_name*
The short name of the action. This is the value specified in the ACTION= attribute of MarkNote and ACTIONS= attribute of MARKLIST.

action title
This is the full name of the action. This is the value presented in the generated marked note list.

Usage

Use MarkAction to define actions that can be associated with marked notes. You can define as many actions as you need to. Any mark action can be used with any mark class.

See *IBM ID Doc User's Guide* for more information on using marked notes.

Examples

```
<prolog>
  ⋮
<PropertyDefinition>
  ⋮
<MarkDesc>
<MarkClass name=changes>Changes
<MarkAction name=new>New
</MarkDesc>
  ⋮
</PropertyDefinition>
  ⋮
</Prolog>
```

SGML Definition

```
<!ELEMENT  MarkAction       - O (%DataPool;)* >
<!ATTLIST  MarkAction
    name             NAME         #REQUIRED
    %Common.Atts;
>
```

MarkClass (marked note class definition)

MarkClass (marked note class definition)

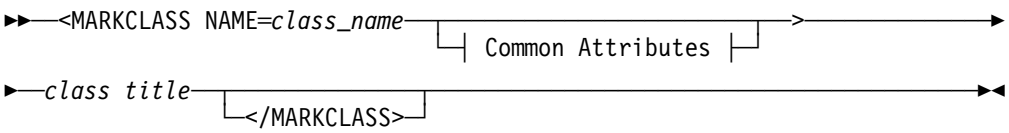
Purpose

Defines a marked note class within MarkDesc.

Valid Contexts

MarkDesc.

Syntax



Attributes and Contained Elements

NAME=*class_name*
The short name of the class. This is the value specified in the CLASS= attribute of MarkNote and CLASSES= attribute of MARKLIST.

class title
This is the full name of the class. This is the value presented in the generated marked note list.

Usage

Use MarkClass to define a class of note. You can define as many classes as you need to. Any mark action can be used with any mark class.

See *IBM ID Doc User's Guide* for more information on using marked notes.

Examples

```
<prolog>
  ⋮
<PropertyDefinition>
  ⋮
<MarkDesc>
<MarkClass name=changes>Changes
<MarkAction name=new>New
</MarkDesc>
  ⋮
</PropertyDefinition>
  ⋮
</Prolog>
```

SGML Definition

```
<!ELEMENT  MarkClass        - O (%DataPool;)* >
<!ATTLIST  MarkClass
    Name             NAME         #REQUIRED
    %Common.Atts;
>
```

MarkDesc (mark description)

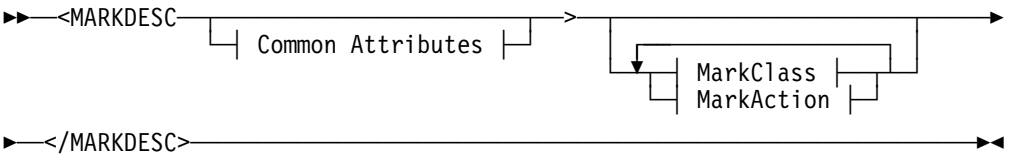
Purpose

Contains MarkClass and MarkAction elements for use with marked notes. MarkClass is valid only within the PropertyDefinitions element.

Valid Contexts

PropertyDefinitions.

Syntax



Attributes and Contained Elements

MarkClass
Defines a marked note class.

MarkAction
Defines a marked note action.

Usage

Use MarkDesc to define the classes and actions that are valid for your marked notes.

See *IBM ID Doc User's Guide* for more information on using marked notes.

Examples

```
<prolog>
  ⋮
<PropertyDefinitions>
  ⋮
<MarkDesc>
<MarkClass name=changes>Changes
<MarkAction name=new>New
</MarkDesc>
  ⋮
</PropertyDefinitions>
  ⋮
</Prolog>
```

SGML Definition

```
<!ELEMENT  MarkDesc         - - (MarkClass | MarkAction)* >
<!ATTLIST  MarkDesc
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

MarkList (marked note list)

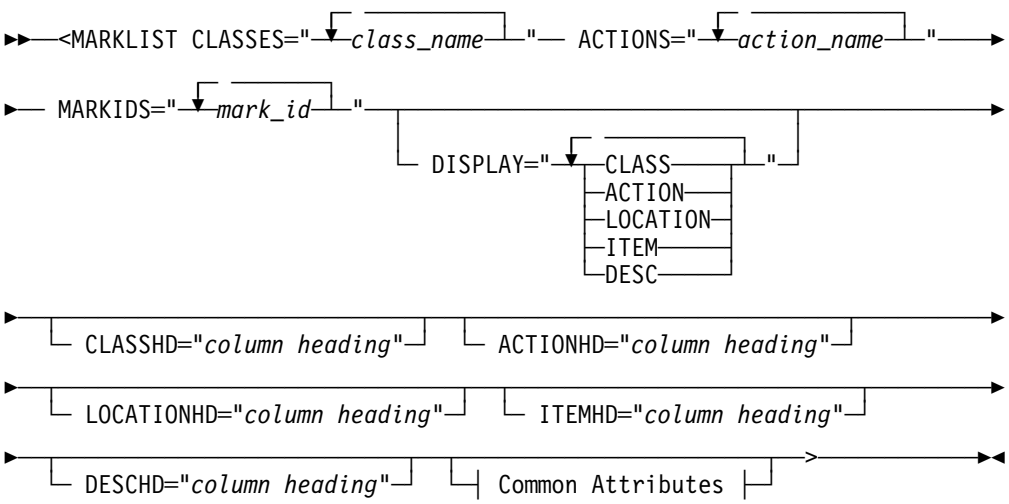
Purpose

MarkList generates a list of marked notes at the point in the document where the MarkList element is specified. Only those notes that are a member of the collections specified and have the specified class and action values will be presented in the list.

Valid Contexts

ParaLevelElements

Syntax



Attributes and Contained Elements

CLASSES=*class values*
Defines the mark class or classes to include in this list. Only notes with a specified class will be presented.

ACTIONS=*action values*
Defines the action or actions to include in this list. Only notes with a specified action will be presented.

MARKIDS=*mark IDs*
Defines which collections to search for notes to present.

DISPLAY=*display items*
Defines the elements to display in the generated listing. You can choose one or more items from the group, although you can't choose any item more than once. The items will be displayed in the list in the order they are specified on the DISPLAY attribute.

LOCATION is the location in the document of the marked note, either a page number or its online equivalent as for cross references.

DESC is the marked note content itself.

CLASSHD=*column heading*
ACTIONND=*column heading*
ITEMHD=*column heading*
LOCATIONHD=*column heading*
DESCHD=*column heading*
Defines the heading text associated with each item in the generated list. These values override the default headings defined in the document style.

Usage

Use marked notes to collect information about the content of your document such as notes about changes, notes to yourself as an author, references to specific locations as navigation aides to readers. Marked notes are collected by BookMaster and then presented in marked note lists as defined by the MARKLIST element.

For example, you can automatically create summary of changes by using marked notes to record changes within the body of your document and then using a MarkList to generate a list of those notes.

Examples

```
<div>Summary of Changes
<marklist markid=v1r2changes class=changes action="new changed deleted">
```

SGML Definition

MarkNote (marked note)

```
<!ELEMENT  MarkList         - O (Title?,
                                 (nmquery | (%InfoUnitPool;)*)
                                )
>
<!ATTLIST  MarkList
    ID               NAME         #IMPLIED
    Classes          NAMES        #REQUIRED
    Actions          NAMES        #REQUIRED
    Display          (Class | Action | Location | Item | Desc)
                                  #IMPLIED
    ClassHD          CDATA        #IMPLIED --Default def'd in doc style--
    ActionHD         CDATA        #IMPLIED --Default def'd in doc style--
    ItemHD           CDATA        #IMPLIED --Default def'd in doc style--
    LocationHD       CDATA        #IMPLIED --Default def'd in doc style--
    DescHD           CDATA        #IMPLIED --Default def'd in doc style--
    MarkIDs          NAMES        #REQUIRED   -- IDs of MARK elements --
    %Common.Atts.Base;

    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED
                          "MarkIDs Mark &Common.RefType"
    InfoMaster       NAME         #FIXED    "QueryLink"
>
```

MarkNote (marked note)

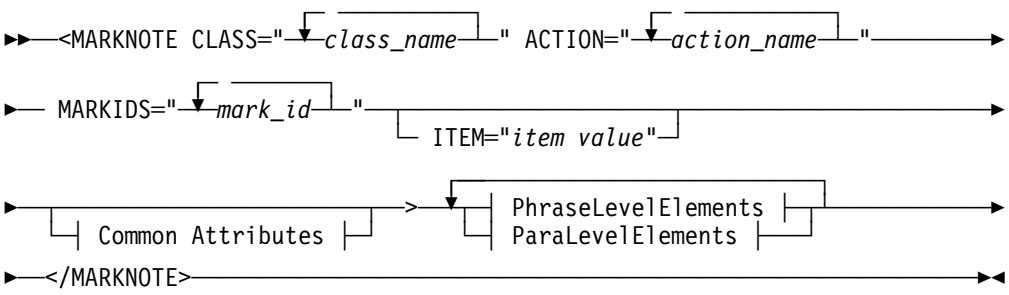
Purpose

Contains a marked note. Marked notes are specialized annotation elements that have specialized properties for associating them with problem and change tracking information, such as change requests, problem numbers, and the like.

Valid Contexts

ParaLevelElements

Syntax



Attributes and Contained Elements

CLASS=*class values*
Defines the mark class or classes to which this marked note belongs.

ACTION=*action values*
Defines the action or actions associated with this particular note.

MARKIDS=*mark IDs*
Contains the IDs of one or more MARK elements. MARK elements define separate collections of marked notes.

ITEM=*item value*
Defines an identifying label for the note, such as a message number, function name, or error report. ITEM= lets you distinquish among marks of the same class and action. You can also use ITEM= to closely associate a note with things in your document that have unique identifiers such as message numbers.

PhraseLevelElements
ParaLevelElements
A marked note can contain phrase- or paragraph-level elements.

Usage

Use marked notes to collect information about the content of your document such as notes about changes, notes to yourself as an author, references to specific locations as navigation aides to readers. Marked notes are collected by BookMaster and then presented in marked note lists as defined by the MarkList element.

For example, you can automatically create summary of changes by using marked notes to record changes within the body of your document and then using a MarkList to generate a list of those notes.

Examples

```
<div>This is new and changed
<marknote markid=v1r2changes class=changes action=new>
<p>New information added about X
</marknote>
  ⋮
```

SGML Definition

```
<!ELEMENT  MarkedNote       - - (%InfoPool;)* >
<!ATTLIST  MarkedNote
    Class            NAMES        #REQUIRED
    Action           NAMES        #REQUIRED
    MarkIDs          NAMES        #REQUIRED   -- IDs of MARK elements --
    Item             CDATA        #IMPLIED
    %Common.Atts.NoRefType.NoClass;

    RefType          CDATA        #FIXED
                          "MarkIDs Mark &Common.RefType"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>
```

MD (deleted phrase)

Purpose

The MD element identifies a phrase which, while not actually deleted, may be presented or otherwise treated as if it were. MD is valid wherever PhraseLevelElements is valid.

Valid Contexts

PhraseLevelElements.

Syntax

Attributes and Contained Elements

REVISION=*revision_id*

The **ID=** value of the Revision element to which this MD element corresponds.

Usage

The MD element is used to identify data that has been deleted in a revision. It is a shorthand for `<phrase revstatus=deleted revision=revision_id>...</phrase>` . It is associated with a specific revision using its Revision= attribute.

MD may contain any other PhraseLevelElements elements. The MD element may not be used to mark data that contains paragraphs or division elements. In these cases, the revision marking capability provided for those elements must be used to indicate the revision level and deletion status of the data.

Examples

```
The following data is deleted in revision 3:
<md revision=rev3>This data is deleted</md>,
as you can clearly see.
```

SGML Definition

```
<!ELEMENT  MD              - - (%DataPool;)*      -- Marked deletion -->

<!ATTLIST  MD

--  Revision         NAME         #REQUIRED --
--  It would be desirable for Revision to be required but since it is --
--  already specified in %Common.Atts;, it is not coded here. --

    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "MarkedDeletion"
```

>

Message (message or code description)

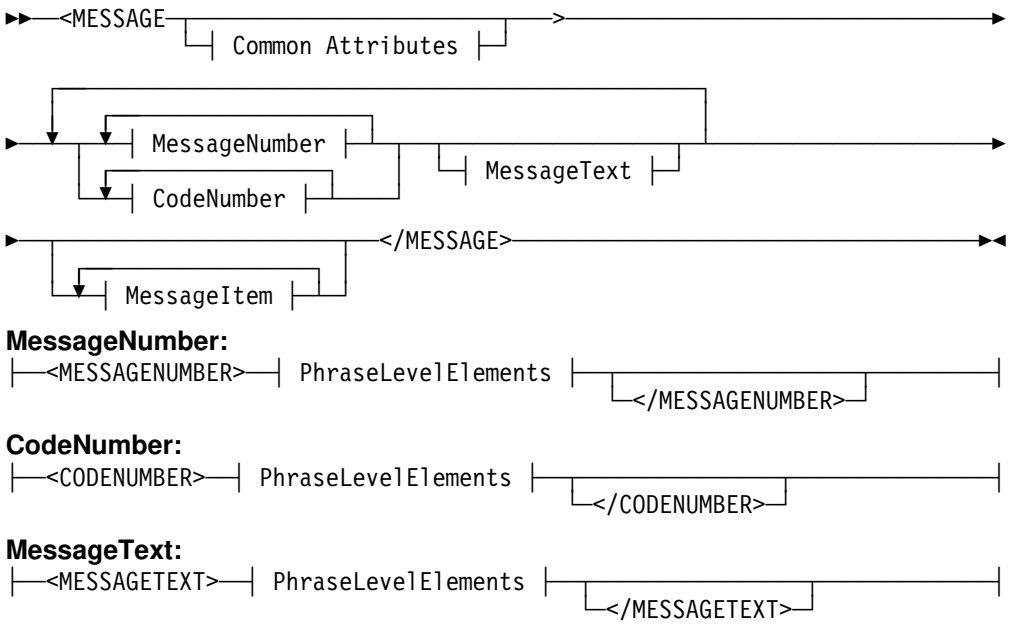
Purpose

Contains a message or code and its description within a MessageList.

Valid Contexts

MessageItem.

Syntax



Attributes and Contained Elements

MESSAGENUMBER
CODENUMBER
The number of the message or code being described.

MESSAGETEST
The text of the message.

MESSAGEITEM
One or more message description items.

Usage

Use the Message element to describe one or more messages or codes. You can use a single Message element to describe multiple messages or codes. You can specify just a message or code number, just a message text, or both. You can also specify a set of number and text pairs.

Examples

```
<MessageList>
<message><messagenumber>IMX001W
<messagetext>Class name &ssq;<mv>class</mv>&ssq; for element
&ssq;<mv>element_name</mv>&ssq; has not been defined
on a ClassDef element
<MessageItem class=explanation>
<p>The element at the point indicated by message <msgno>IMX002I</>
has a Class= attribute specified but the class name has not been
defined by a ClassDef element.
Element Class= values must be defined using a ClassDef element
before it can be used on an element.
The ClassDef must occur in the Prolog or DivProlog section of
one of the direct ancestors the element using the class value.
<MessageItem class=UserRepsonse>
<p>Check the Class= value to make sure it is specified correctly.
If the Class= value is defined on a ClassDef element within the
document, make sure it is within a direct ancestor of
the element that caused this message.
If the Class= value is not defined on a ClassDef element,
create a ClassDef element.
See <hdref refid=classdef docid=IBMIDDOCreference> for
more information on ClassDef.
</message>
  ⋮
</MessageList>
```

SGML Definition

```
<!ELEMENT  Message          - O (((CodeNumber* | MessageNumber*),
                                  MessageText?)*,
                                 MessageItem*)
                                                  -- Message Module --
>
<!ATTLIST  Message
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
>
```

MessageItem (message description item)

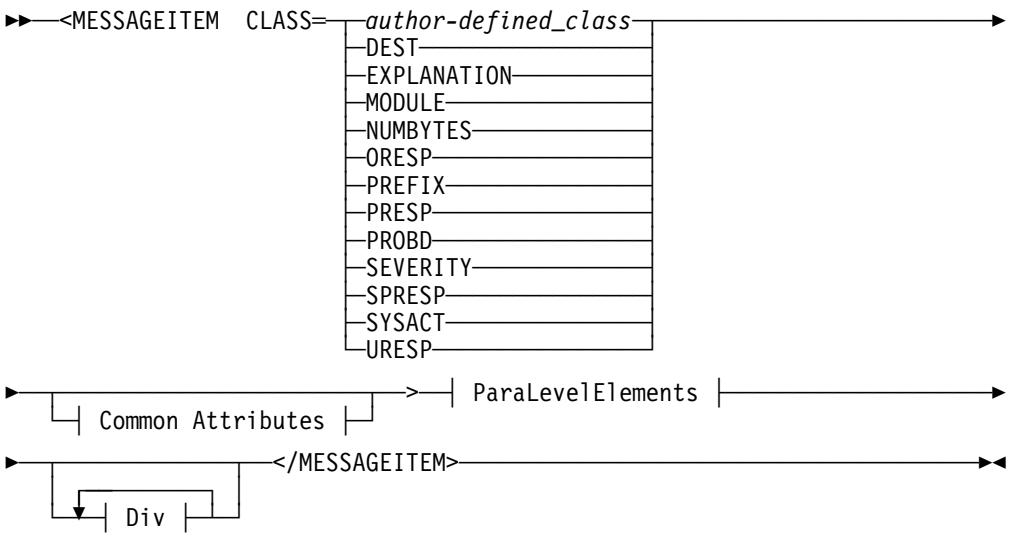
Purpose

Contains part of the description of a message or code within a Message element.

Valid Contexts

Message.

Syntax



Attributes and Contained Elements

CLASS=*classname*

Indicates the class of the MessageItem. The class values are defined for a given modular information class on the MessageList or MessageListDef element.

The class values shown in the syntax definition are pre-defined; you can also define your own classes if you need to. The defined class values are:

author-defined_class
An author-defined class value. Must be defined using a MessageItemDef element.

DEST
Message destination.

EXPLANATION
The explanation of the message.

MODULE
Issuing module.

NUMBYTES
Number of error bytes.

ORESP
Operator response.

MessageItem (message description item)

PREFIX
Prefix to the message.

PRESP
Programmer response.

PROBD
Problem determination information.

SEVERITY
The severity of the message.

SPRESP
System programmer response.

SYSACT
System action.

URESP
User reponse.

Usage

Examples

```
<MessageList>
<message><messagennumber>IMX001W
<messagetext>Class name &ssq;<mv>class</mv>&ssq; for element
&ssq;<mv>element_name</mv>&ssq; has not been defined
on a ClassDef element
<MessageItem class=explanation>
<p>The element at the point indicated by message <msgno>IMX002I</>
has a Class= attribute specified but the class name has not been
defined by a ClassDef element.
Element Class= values must be defined using a ClassDef element
before it can be used on an element.
The ClassDef must occur in the Prolog or DivProlog section of
one of the direct ancestors the element using the class value.
<MessageItem class=UserRepsonse>
<p>Check the Class= value to make sure it is specified correctly.
If the Class= value is defined on a ClassDef element within the
document, make sure it is within a direct ancestor of
the element that caused this message.
If the Class= value is not defined on a ClassDef element,
create a ClassDef element.
See <hdref refid=classdef docid=IBMIDDOCreference> for
more information on ClassDef.
</message>
  ⋮
</MessageList>
```

SGML Definition

```
<!ELEMENT  MessageItem      - O ((%InfoUnitPool;)*, Div)
                                          -- Message Description Item --
>
<!ATTLIST  MessageItem
    Class            CDATA        #REQUIRED
                    -- Valid values defined on MessageItemDef element --
    %Common.Atts.NoClass;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MessageItem"
>
```

MessageList
List of message descriptions

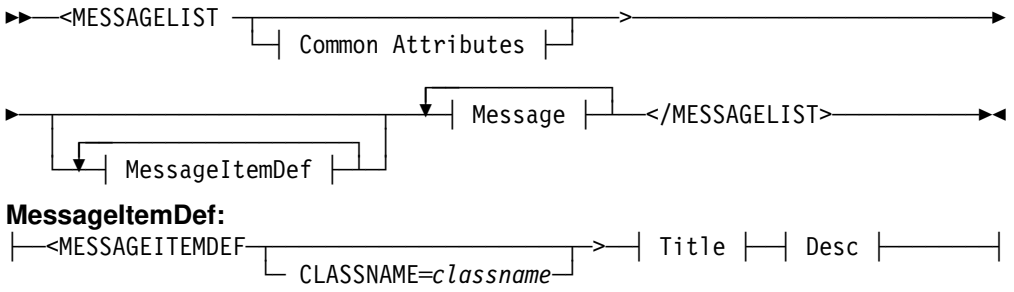
Purpose

Contains the description of one or more messages issued by a computer system. Messages may be represented by a code number, code number and message text, or just message text.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

MessageItemDef
Defines or modifies classes of message description items valid for this MessageList. Each MessageItemDef contains a Title and a Desc.

CLASS=*classname*
The message item class to which this message item definition applies. See “MessageItem (message description item)” on page 206 for complete description of the message item classes.

Title
The Title is the title to be generated for a given class of ModuleItem.

Desc
The Desc describes the purpose or meaning of a given ModuleItem class.

Message Element
A message list contains one or more Message elements.

Usage

Use MessageList to contain descriptions of messages and codes {ssued by computer systems. A MessageList contains one or more Message elements, which contain the message or code and any explanatory text associated with it.

Examples

```
<MessageList>
<message><messagenumber>IMX001W
<messagetext>Class name &ssq;<mv>class</mv>&ssq; for element
&ssq;<mv>element_name</mv>&ssq; has not been defined
on a ClassDef element
<MessageItem class=explanation>
<p>The element at the point indicated by message <msgno>IMX002I</>
has a Class= attribute specified but the class name has not been
defined by a ClassDef element.
Element Class= values must be defined using a ClassDef element
before it can be used on an element.
The ClassDef must occur in the Prolog or DivProlog section of
one of the direct ancestors the element using the class value.
<MessageItem class=UserRepsonse>
<p>Check the Class= value to make sure it is specified correctly.
If the Class= value is defined on a ClassDef element within the
document, make sure it is within a direct ancestor of
the element that caused this message.
If the Class= value is not defined on a ClassDef element,
create a ClassDef element.
See <hdref refid=classdef docid=IBMIDDOCreference> for
more information on ClassDef.
</message>
  ⋮
</MessageList>
```

SGML Definition

```
<!ELEMENT  MessageList      - - (MessageItemDef*, Message+) >
<!ATTLIST  MessageList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

ModularInfoDef (modular information property definition)

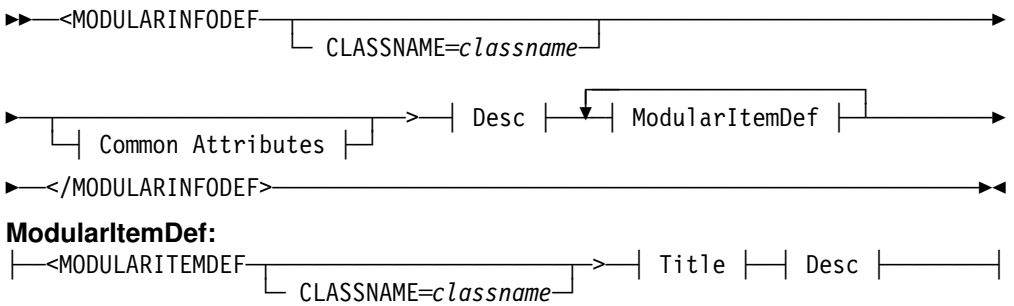
Purpose

Defines a set of modular information properties. Contains the definition of the ModuleItem properties.

Valid Contexts

ProperyDefinitions.

Syntax



Attributes and Contained Elements

CLASSNAME=*classname*
Defines the name of this class of modular information.

Desc
An optional text description of this ModularInformation class. This is intended to provide documentation about why a certain class has been defined.

ModularItemDef
Defines the classes of module description items valid for this class of ModularInformation. Each ModularItemDef contains a Title and a Desc. The Title is the title to be generated for a given class of ModuleItem. The Desc describes the purpose or meaning of a given ModuleItem class.

Usage

Use ModularInfoDef to define a class of modular information specific to your information. You can apply specific presentation styles to a given class of modular information.

Examples

```
<ModularInfoDef classname=onlinepanel>
<desc>Defines the elements of a single online panel.
<ModularItemDef classname=titlebar>
<title>Title Bar
<desc>Contains a definition of the panel's title bar.
</ModularInfoDef>
```

SGML Definition

```
<!ELEMENT  ModularInfoDef   - O (Desc, ModularItemDef+)
                                    -- Module Information Definition --
>
<!ATTLIST  ModularInfoDef
    ClassName        NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "PropertyDefinition"
>
```

ModularInformation

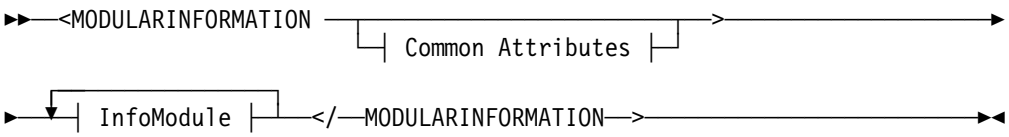
Purpose

Contains a set of information modules for a single class of modular information.

Valid Contexts

ParaLevelElements

Syntax



Attributes and Contained Elements

InfoModule
One or more InfoModule elements.

Usage

Use ModularInformation to contain sets of information modules describing a particular type of thing specific to your document. For example, in a document describing a complex program, you might define a class of information module for describing data structures.

Information modules can also be organized within object libraries and used by reference from within a modular information section using the common ContentRef= attribute.

The default presentation style for modular information is as normal divisions, but you can define other presentation styles. For example, you may want to present each module as a row in a table for quick reference.

To use information modules, you must define an information module class using the ModularInfoDef element. You can specify the modular information class on the ModularInformation element, in which case it applies to all contained InfoModule elements, or you can specify the appropriate class on each InfoModule element directly. For example, you might have defined several related classes of informa-

tion module that you want to group into a single ModularInformation group for presentation. In this case, you would need to specify the class for each module directly, rather than specifying a global class on the ModularInformation element.

Examples

```
<ModularInformation>
<InfoModule id=fubar>
<ModuleName>FUBAR command
<ModuleDesc>Slices, dices, makes julien fries
<ModuleItem class=purpose>
<p>FUBAR performs the slicing and dicing function.
<ModuleItem class=syntax>
<syntax>Fubar command
</syntax>
</InfoModule>
  ⋮
</ModularInformation>
```

SGML Definition

```
<!ELEMENT  ModularInformation
                                - - (InfoModule+)
>
<!ATTLIST  ModularInformation
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleList"
>
```

ModuleItem (module description item)

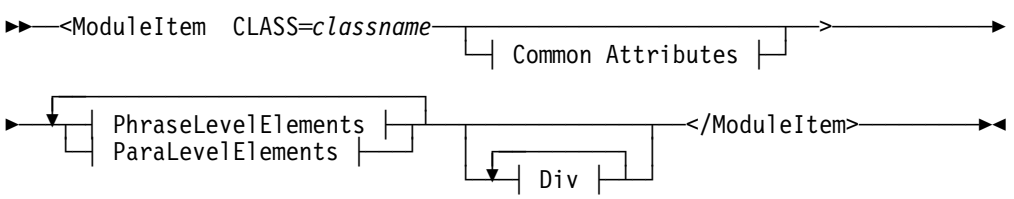
Purpose

Contains a description of one aspect of an information module.

Valid Contexts

InfoModule.

Syntax



Attributes and Contained Elements

CLASS=*classname*

Indicates the class of the ModuleItem. The class values define what type of information each ModuleItem contains, such as format, examples, purpose, and comments. The class values are defined for a given modular information class on the ModularInfoDef element.

PhraseLevelElements
ParaLevelElements

ModuleItems can contain just phrase-level elements or paragraph-level elements as well.

Div

As for DivBody, ModuleItems can contain divisions following any phrase- or paragraph-level elements.

Usage

The ModuleItems elements within an InfoModule define the structure of the description by subdividing it into subsections with meaningful names.

For each ModuleItem element, the title text is defined separately on a ModularInfoDef element or in the document style definition. Each ModuleItem class can have a different generated title or no title. The default presentation style for ModuleItems elements is as normal divisions, but other styles can be used as well. For example, you can define a table-like presentation for InfoModules. In that case, the generated ModuleItem title would be used as the column headings for the columns corresponding to each ModuleItem class. Also, through the document style definition, you can choose to suppress the presentation of any ModuleItem class. For example, you may want to only show the format items in a quick reference created from the same source that is used for the full reference document.

ModuleItems can contain divisions.

Examples

```
<ModularInformation>
<InfoModule id=fubar>
<ModuleName>FUBAR command
<ModuleDesc>Slices, dices, makes julien fries
<ModuleItem class=purpose>
<p>FUBAR performs the slicing and dicing function.
<ModuleItem class=syntax>
<syntax>Fubar command
</syntax>
</InfoModule>
  ⋮
</ModularInformation>
```

SGML Definition

```
<!ELEMENT  ModuleItem       - O (%InfoPool;)*          -- Module Item -->
<!ATTLIST  ModuleItem
    Class            CDATA        #REQUIRED
    -- Valid values defined on ModularInfoDef element --
    %Common.Atts.NoClass

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ModuleItem"
>
```

MultiMediaObject (non-text object)

Purpose

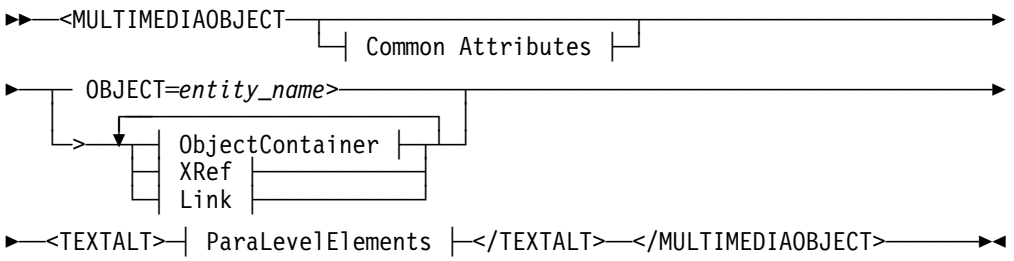
MultiMediaObject links a document to a non-text object, such as an image, vector graphic, or video clip. For print, non-text objects are normally image or vector graphics, such as CGM or TIFF graphics. The types of objects supported is a function of the processing and presentation system you are using.

MultiMediaObject also contains a text description of the object, intended to provide an alternative to the object itself.

Valid Contexts

FigureBody, FigSeg, ParaLevelElements, PhraseLevelElements.

Syntax



Attributes and Contained Elements

OBJECT=*entity_name*

Refers to an entity that represents the non-text object. When the Object= attribute is used, you cannot specify ObjectContainer, XRef, or Link.

ObjectContainer

Contains or refers to a non-text object. Use ObjectContainer when you need to refer to multiple objects within a single MultiMediaObject element.

Xref
Link

Define a hyperlink to the non-text object.

TextAlt

Contains a text description of the multi-media object, to be presented as an alternative to the object itself, primarily for the benefit of the visually disabled.

Usage

Use MultiMediaObject to create a direct reference to a non-text object, such as a graphic or video clip in the body of the document. Depending on the style definition, the object may be integrated with the text. For print, the default and expected presentation style is for the object to be presented in line with the text. The presentation of non-text objects by online presentation systems is a function of the online presentation system, not IBM ID Doc.

When you only need to refer to a single non-text object, you can use the Object= attribute of the MultiMediaObject element to refer directly to the entity that represents the object (for example, an image file). When you need to refer to multiple objects, or want to refer to objects indirectly using a cross reference or hyperlink phrase, you must use ObjectContainer, XRef, or Link elements within MultiMediaObject. For example, you may have a separate multimedia presentation to which you want to link and with which you want to associate a text alternative. You can do this by using Link within MultiMediaObject to refer to the multimedia presentation.

In most cases, the Object= form will be the one to use.

Within MultiMediaObject, you must provide a text description of the non-text object as an alternative to the object for use by the visually impaired and with presentation systems that are not capable of presenting the non-text object. For instance, in the example above where Link is used to refer to a separate multimedia presentation, the text alternative could describe what you would see and could be used in the printed version of the information.

Migration Note

MultiMediaObject replaces the ARTWORK and ARTALT tags. Use MultiMediaObject as you would ARTALT. The presentation attributes on the ARTWORK tag are now notation attributes specified on the entity declaration used to declare the artwork file. See “Notation Attributes” on page 227 for more information on specifying notation attributes.

Examples

```
<MultiMediaObject object=GraphicEntityName>
<textalt>
<p>This describes the graphic with words.
</textalt>
</MultiMediaObject>
```

This example shows the normal case where you are simply including a graphic. The Object= attribute takes the name of the entity that represents the graphic. The next example shows how you could use Link to refer to a multimedia presentation:

```
<LinkDesc id=mmtutorial-start>
<desc>Sets up a link to the multimedia tutorial
for &prodname;, starting at the normal entry point to
<nmlist entity>mmtutorial.entrypoint
</LinkDesc>
  ⋮
<p>If you would like a tutorial on &prodname,
<MultiMediaObject>
<link linkends=mmtutorial-start>click here</link>
<TextAlt>
If you select the text <char>click here</char>,
the &prodname; tutorial is started at
the main entry point.
The main entry point panel contains....
</TextAlt>
</MultiMediaObject>
```

In this example, the LinkDesc element defines the linkage to the multimedia tutorial, allowing reference to it from anywhere using a variety of hyperlinking methods. The use of Link within MultiMediaObject serves two purposes in this example. First, it provides a hyperlink "button" that is integral with the text. Second, it associates the link to the tutorial with a text description so that when the tutorial is not available (such as in the printed version of the information), as description of what you would see can be presented.

SGML Definition

```
<!ELEMENT  MultiMediaObject - O
                     ((ObjectContainer | Link | Xref)*, TextAlt)
>
<!ATTLIST  MultiMediaObject
    Object          ENTITY       #IMPLIED
    %Common.Atts;

    InfoMaster      NAME         #FIXED     "MultiMediaContainer"
>

<!ELEMENT  TextAlt          - O (%InfoPool;)*
>
<!ATTLIST  TextAlt
    %Common.Atts;

    InfoMaster      NAME         #FIXED     "TextAlternative"
>
```

MV (message variable)

Purpose

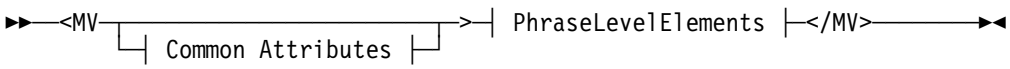
The MV element identifies the variable parts of a message issued by a computer system.

Name (person's name)

Valid Contexts

PhraseLevelElements.

Syntax



Usage

The MV element is used to identify data that is a placeholder for variable data in a message. It may be used either in the message text itself or in the explanation of the message.

The content of this element consists of a name, keyword or other placeholder for data to be substituted by the system when the message in question is presented.

Examples

The **<mv>filename</mv>** string indicates the name of the file which contains the erroneous information.

SGML Definition

```
<!ELEMENT  MV               - - (%DataPool;)*     -- Message Variable -->
<!ATTLIST  MV
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "MessageVariable"
>
```

Name (person's name)

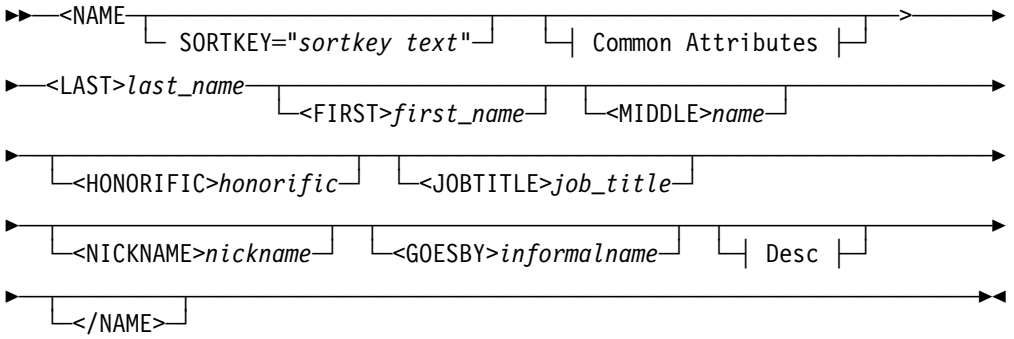
Purpose

Contains a person's name, structured to allow a variety of presentation styles and accurate retrieval of names by automated systems.

Valid Contexts

Person.

Syntax



Attributes and Contained Elements

SORTKEY="*sortkey text***"**
Defines the text string that should be used when sorting the name.

LAST
The person's last name or surname.

FIRST
The person's first name or initial.

MIDDLE
The person's middle name or initial.

HONORIFIC
The honorific by which the person prefers to be known, for example, “Dr.”.

JOBTITLE
The person's job title.

NICKNAME
The person's nickname.

GOESBY
The way the person prefers to be referred to informally.

Usage

The NAME element structures people's names to enable sophisticated presentation of names, for example, automatic sorting of names or use of initials only, retrieval of author information by search and retrieval systems, management of distribute lists and so on.

Only a person's last name is required. The other elements of the name must be specified in the order shown in the syntax diagram.

Names can be given an ID to allow use of a single name definition in many places by reference.

Names are normally used within an Author element but may appear in other contexts.

Examples

```
<name id=drmacro>
  <last>Kimber<first>William<middle>Eliot
  <honorific>Mr.<jobtitle>Information Developer
  <nickname>Dr. Macro<goesby>Eliot
</name>
```

SGML Definition

Nameloc (named location)

Purpose

The Nameloc element associates a local ID with other IDs, using either name list (NMList) or name query (NMQuery) elements. Nameloc can occur within LinkDesc or wherever paragraph-level elements are valid.

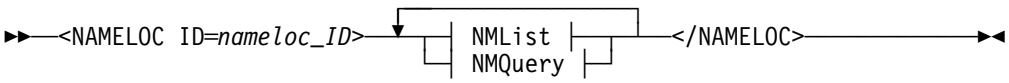
Nameloc is the standard HyTime mechanism for creating indirect hyperlinks.

When used in conjunction with the Link or XRef element, or when used with author-defined anchor roles, Nameloc should be used within a LinkDesc in order to associate a description of the hyperlink with its HyTime definition. When Nameloc is used with the IBM ID Doc-defined anchor roles Annotation, Alternate, PropertySource, or RevisionSpecification, it need not be put inside a LinkDesc element, as the purpose of the Nameloc is defined by the anchor role it's used with.

Valid Contexts

LinkDescriptions, NMlist, NMQuery.

Syntax



Attributes and Contained Elements

ID=*nameloc_ID*

The ID of the Nameloc element. This is the ID refered to from the Refid=, Linkend=, or Linkends= attribute of a hyperlinking element.

NMList

A NMList element that contains the IDs or entity names of the objects being linked to. NMList also takes the Docorsub= attribute used to specify the document or subdocument the IDs are in. See “NMList (list of IDs or entities)” on page 222 for more information on using NMList elements.

NMQuery

A name query element that contains a query that will resolve to the IDs or entity names of the objects being linked to. See “NMQuery (named query)” on page 225 for more information on using NMQuery elements.

Usage

Nameloc is the basic indirection mechanism defined by the HyTime standard and is used with all hyperlinks, regardless of purpose. The indirection provided by Nameloc serves the following purposes:

- Associates a single ID with multiple objects (either other SGML elements or entities)
- Associates a local ID with objects in another document or subdocument.
- Associates a local ID with one or more queries that will locate objects to be linked to.

Nameloc is required when you want to link to multiple objects from a Link or XRef element, for a single anchor role (such as the Alternate role used to link an element to its retrieval alternatives), or when you want to create a cross-document link.

The most common use of Nameloc is to contain a single NMList element used to either create a simple cross-document link or to create a multi-object link. However, Nameloc can contain multiple NMList and NMQuery elements. For example, you could create a link to multiple targets in different documents by using a new NMList for the targets in each different document, or you could combine a NMList with a NMQuery to link to some objects explicitly and others implicitly via the query.

Note: It is expected that in most cases, Nameloc elements will be created by the authoring system automatically, shielding you from the details of specifying HyTime-compliant hyperlinks. Nevertheless, it is important to understand how hyperlinks are constructed in HyTime and IBMIDDoc so that you can make the best use of the features of your processing and presentation system.

Examples

The first example shows a simple Nameloc used to create a cross-document link. Note that the entity named on the Docname= attribute has been declared as a data entity in the document's SGML prolog.

```
<linkdescriptions>
<linkdesc>
<desc>A simple cross-document link.
</desc>
  <nameloc id=namelocref>
    <nmlist docname="iddocref">nameloc</nmlist>
  </nameloc>
</linkdesc>
</linkdescriptions>
  ⋮
<p>The <link linkend=namelocref>Nameloc</link> element is
the workhorse of the HyTime architecture.
```

In this example, the Nameloc element, which is within a LinkDesc element, contains a single NMList element. The NMList element associates the target ID (in this case the ID of the reference entry for Nameloc) with the document it is in (the document represented by the entity named 'iddocref'). The Link element itself points to the Nameloc element, which then serves to locate the actual target, the element with the ID 'nameloc' in the document 'iddocref'.

The next example shows how to use Nameloc to link to multiple targets, in this case, multiple alternates for a single element.

```
<p id=para-1 linkends="para-1-alts"
          anchorroles="linksource alternate"
          HyTime=ilink>
<nameloc id=para-1-alts>
 <nmlist>para-1-French para-1-Spanish</nmlist>
</nameloc>
This is the base version of this paragraph.
</p>
```

In this example, the paragraph uses the Alternate anchor role to associate itself with alternate versions (presumably held somewhere else, such as in an ObjectLibrary). Since there are two versions, a Nameloc must be used to associate the single Linkends= ID, which corresponds to the Alternate anchor role, with multiple targets. Thus the NMList within in the Nameloc lists two IDs, para-1-French and para-1-Spanish.

Note that in this example, the Nameloc is put right into the paragraph, since it only applies to this paragraph and does not need the additional description provided by LinkDesc.

SGML Definition

```
<!ELEMENT  nameloc          - O (nmquery | nmlist)*
                                        -- HyTime nameloc element --
>
<!ATTLIST  nameloc
    ID              NAME         #REQUIRED

            -- HyTime attributes: --
    HyTime          NAME         #FIXED    namloc
    ordering        (ordered)    ordered
              -- Up to presentation system to define meaning of order --
    set             (set)        set                 -- Ignore dupes --
    aggloc          (nagg)       nagg               -- Not aggregate --
    spanloc         (nspan)      nspan                   -- No spans --
    limsort         (nlimsort)   nlimsort        -- No limit sorting --
>
```

NMList (list of IDs or entities)

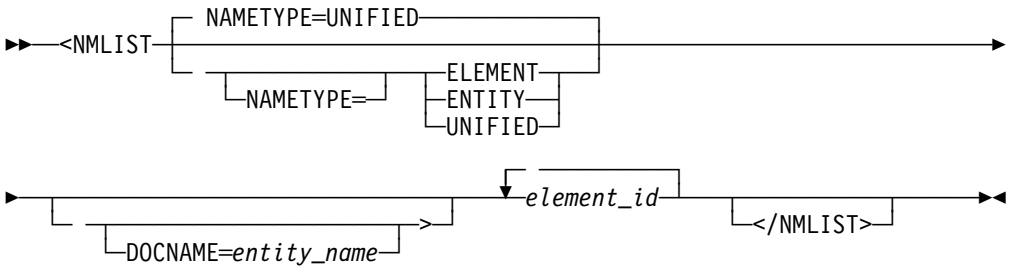
Purpose

Name list (NMList) is used within a Nameloc element to associate the Nameloc element's ID with the IDs of other elements or the IDs of other elements in other documents.

Valid Contexts

Nameloc.

Syntax



Attributes and Contained Elements

NAMETYPE=UNIFIED | ENTITY | ELEMENT

Indicates whether the contained names are element IDs or entity names. The default name type is 'UNIFIED', indicating that IDs and entity names together must be unique within a document, and that entities can be refered to with what would otherwise be an ID reference.

ENTITY indicates that the names listed in the content of the NMList element are declared entity names.

ELEMENT indicates that the names listed in the content of the NMList element are element IDs.

element_IDs

One or more element IDs, representing the IDs of the elements to be located. When the unified name space option is in effect (either because the HyTime unmspace= attribute was specified as Unified on the document element or because the Nametype= value is Unified), the IDs can also be the names of entities to be located.

DOCNAME=*entity_name*

The name of the document or subdocument entity that contains the IDs or declares the entities listed in the content of the NMList element. The entity named must be declared in the current document.

Note: Docname= is the HyTime docorsub= attribute

Usage

NMList is used within Nameloc to hold the names (IDs or entity names) of the objects to be located, either SGML elements or entities. When the objects to be located are in the same document as the NMList element, you simply list their IDs or names in the content of the NMList. When the objects are in another document or subdocument, you must also specify the Docname= attribute to indicate the document that contains the target objects.

You can use multiple NMList elements within a single Nameloc. For example, you may need to list elements both in the current document and in other documents. You need to specify a new NMList for each different document that contains target objects.

Examples

The first example shows a simple NMList used to create a cross-document link. Note that the entity named on the Docname= attribute has been declared as a data entity in the document's SGML prolog.

```
<linkdescriptions>
<linkdesc>
<desc>A simple cross-document link.
</desc>
<nmlist id=nmlistref>
<nmlist docname="iddocref">nmlist</nmlist>
</nmlist>
</linkdesc>
</linkdescriptions>
  ⋮
<p>The <link linkend=nmlistref>NMList</link> element is
the workhorse of the HyTime architecture.
```

In this example, the NMList element, which is within a LinkDesc element, contains a single NMList element. The NMList element associates the target ID (in this case the ID of the reference entry for NMList) with the document it is in (the document represented by the entity named 'iddocref'). The Link element itself points to the NMList element, which then serves to locate the actual target, the element with the ID 'nmlist' in the document 'iddocref'.

The next example shows how to use NMList to link to multiple targets, in this case, multiple alternates for a single element.

```
<p id=para-1 linkends="para-1-alts"
anchorroles="linksource alternate"
HyTime=ilink>
<nmlist id=para-1-alts>
<nmlist>para-1-French para-1-Spanish</nmlist>
</nmlist>
This is the base version of this paragraph.
</p>
```

In this example, the paragraph uses the Alternate anchor role to associate itself with alternate versions (presumably held somewhere else, such as in an ObjectLibrary). Since there are two versions, a NMList must be used to associate the single Linkends= ID, which corresponds to the Alternate anchor role, with multiple targets. Thus the NMList within in the NMList lists two IDs, para-1-French and para-1-Spanish.

Note that in this example, the NMList is put right into the paragraph, since it only applies to this paragraph and does not need the additional description provided by LinkDesc.

SGML Definition

```
<!ELEMENT  nmlist           - O (#PCDATA)         -- HyTime name list -->
<!ATTLIST  nmlist
    -- HyTime attributes: --
    HyTime           NAME         #FIXED    nmlist
    nametype         (entity | element)
                                  element     --Unified not supported --
    obnames          (nobnames)   nobnames
    docname          ENTITY       #IMPLIED
    dtdorlpd         NAMES        #IMPLIED
    HyNames          CDATA        #FIXED    "docorsub docname"
>
```

NMQuery (named query)

Purpose

The NMQuery element contains a query in some supported query notation. HyQ is the recommended query notation.

Query can be used within the content of elements that can generate their content by query, such as TOC and Index. NMQuery can also be used within LinkDescriptions to define queries for use with the NMQuery element within LinkDesc.

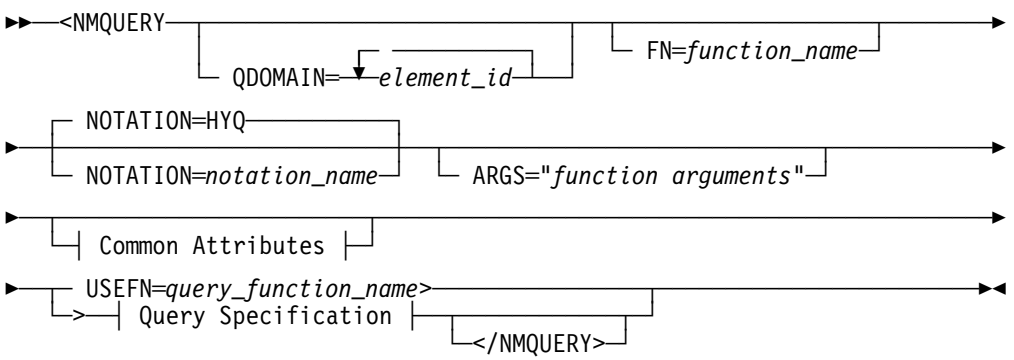
The use of the NMQuery element to define new queries (as opposed to using pre-defined queries) is intended for specialists familiar with the query notation being used (e.g., HyQ). The IBM ID Doc application pre-defines a number of queries designed to meet the normal needs of IBM ID Doc users. These queries can be used directly without the need to understand the query notation.

See Appendix C, “Predefined HyQ Query Functions for IBM ID Doc” on page 339 for a description of the pre-defined queries defined by IBM ID Doc. Your processing system or system administrator may provide other pre-defined queries.

Valid Contexts

LinkDescriptions, Nameloc, FigureList, TOC, Index, TableList, BibList, IBMSafety, Safety, SOA.

Syntax



Attributes and Contained Elements

qdomain=*element-ids*
The starting point for the query. The query domain is normally implied by the application depending on the specific type of query being performed.

FN=*function-name*
Assigns a query function name to the contained query. The named query can then be used by reference from other NMQuery elements or from within other queries, if that function is supported. For HyQ queries, named queries can be used via the UseQ() function.

USEFN=*function-name*
The name of a previously-defined function to use for this query. When UseFN= is specified, it is an error specify any content or the NMQuery end tag.

ARGS=*function-arguments*
Any argument values to be passed to the function named by UseFN=.

NOTATION=*query_notation*
Defines the notation the query is in. The default notation is 'HyQ', the HyTime query language.

Query Specification
A query in some query language, such as HyQ. IBM ID Doc recommends HyQ as the query language implementations should use, but specific IBM ID Doc implementations are free to use any query notation they wish.

Usage

Within the context of a query link-type element, such as TOC, Figurelist, or Index, use NMQuery to refer to a pre-defined query or to contain a new query to access the data you want to have presented in place of the query link element.

Within the context of LinkDescriptions, use NMQuery to define query functions for use by reference from other queries.

Examples

The query below is used within FigureList to create a list of only those figures that have a defined class of 'ExplodedDiagram'. The All_Figures_of_Class_X is a pre-defined query defined by the IBM ID Doc application. It takes one argument, the class name used to select the figures:

```
<figlist>
<nmquery usefn="All_Figures_of_Class_X" args="ExplodedDiagram">
</figlist>
```

SGML Definition

```
<!ELEMENT  nmquery          - O (#PCDATA)       -- HyTime name query -->
<!ATTLIST  nmquery
    fn       -- Function name --
             -- Constraint: unique in element type --
                    NAME     #IMPLIED  -- Default: none --
    usefn    -- Fn of another function to which to pass
                the args of this one --
                    NAME     #CONREF   -- Default: none --
    args     -- Argument string passed to the function
                (if supported by function language) --
                    CDATA    #IMPLIED  -- Default: none --

    ID               NAME         #IMPLIED
    notation         NAME         #IMPLIED  -- Default is HyQ --
    -- HyTime attributes: --
    HyTime           NAME         #FIXED    nmquery
    qdomain          NAMES        #IMPLIED
    qcontext         NAME         #IMPLIED
>
```

Notation Attributes

Purpose

Notation attributes primarily define the presentation aspects of NDATA entities, but can also be used with SDATA and CDATA entities.

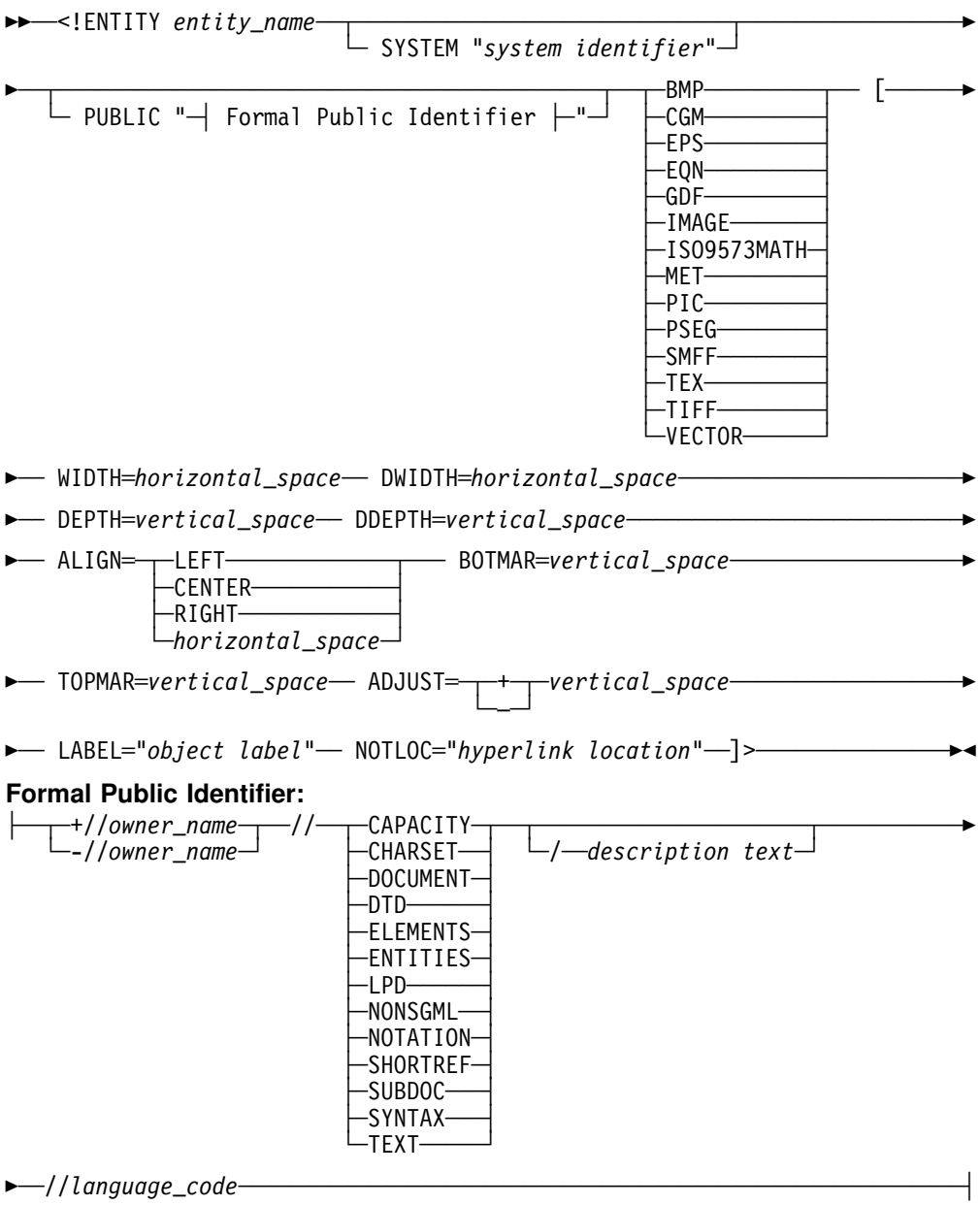
Valid Contexts

Specified on declarations of graphic data entities.

Syntax

Attributes for Graphic Notations

Notation Attributes



Attributes and Contained Elements

WIDTH=*horizontal space*
Defines the presentation width of the entity.

DWIDTH=*horizontal space*
Defines the presentation width for character-based display devices.

DEPTH=*vertical space*
Defines the presentation depth of the entity.

DDEPTH=*vertical space*
Defines the presentation depth of the entity for character-based display devices.

ALIGN=*align value*
Defines how the entity should be aligned within the presentation space.

TOPMAR=*vertical space*
Defines the amount of white space to put above the entity.

BOTMAR=*vertical space*
Defines the amount of white space to put below the entity.

ADJUST=*vertical space*
Defines how the entity should be adjusted relative to the current baseline of the output line when the entity is presented inline in the text.

LABEL="*object label***"**
A text label to be used in place of the object when it can't be presented or to use as a hyperlink button.

NOTLOC="*hyperlink anchor location***"**
Application or notation-specific specification of one or more hyperlink anchors within the data entity. See the description of the NotLoc element for details on notation locations.

Usage

Use notation attributes to define the presentation aspects of a given non-SGML (ndata) entity. You can define different behaviors for the same object by defining multiple entities with different sets of notation attributes.

Examples

```
<!-- The graphic 'mygraphic' is an encapsulated postscript image -->
<!ENTITY mygraphic SYSTEM "mygraphic" eps
         [width="4 inches" depth="5 inches" label="my graphic"]
>
```

SGML Definition

```
<!--GraphicNotation.Atts defines notation attributes for use with
     all graphic notations.
-->

<!ATTLIST  #NOTATION (%Graphic.Notations;)
    WIDTH            CDATA        #IMPLIED -- Width of graphic --
    DWidth           CDATA        #IMPLIED -- Width for online display --
    DEPTH            CDATA        #IMPLIED -- Depth of graphic --
    DDepth           CDATA        #IMPLIED -- Depth for online display --
    ALIGN            CDATA        #IMPLIED -- Horizontal alignment --
    BOTMAR           CDATA        #IMPLIED -- Margin below graphic --
    TOPMAR           CDATA        #IMPLIED -- Margin above graphic --
    ADJUST           CDATA        #IMPLIED -- Adjust rel to baseline --
    LABEL            CDATA        #IMPLIED -- Label for graphic --
>
```

Note

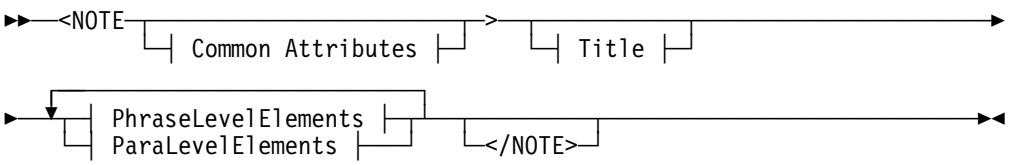
Purpose

Contains a note.

Valid Contexts

ParaLevelELements.

Syntax



Attributes and Contained Elements

Title

In the default style, the generated English language title for Note is 'Note:'. When specified, the Title replaces any generated title.

Migration Note

Note replaces the NT and NOTE elements.

PhraseLevelElements
ParaLevelElements

Notes have the same content model as paragraphs, except that Note can contain paragraphs

Usage

Use Note to create a note of one or more paragraphs.

Examples

Simple example:

```
<note>This is a note about something.
```

Example with title:

```
<Note><title>Note title</title>
Note content
```

SGML Definition

```
<!ELEMENT  Note             - O (Title?, NoteBody) >
<!ATTLIST  Note
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
>

<!ELEMENT  NoteBody         O O (%InfoPool.NoNote;)* >
<!ATTLIST  NoteBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ParagraphBody"
>
```

NoteList (ordered note list)

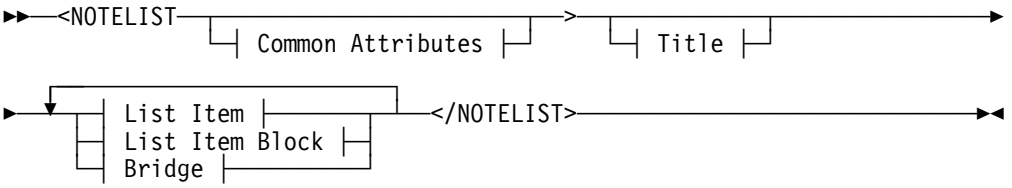
Purpose

Contains a list of notes.

Valid Contexts

ParaLevelElements

Syntax



Attributes and Contained Elements

Title
Use Title to define an alternate title for the note list. The default English language title for Note List is 'Notes:'.

List Item
A list item.

List Item Block
Organizes list items within the note list.

Bridge
A Bridge element, normally used to define a transition or linkage between list item blocks or list items.

Usage

Note lists are valid anywhere paragraphs are valid. Paragraphs can contain lists, so you must explicitly end a paragraph before a list if you want the list to be outside the paragraph.

Examples

```
<notelist>Dietary Notes
<li>Bread contains yeast
<li>Milk contains water and protein
<li>Peanut butter contains salt
</notelist>
```

SGML Definition

```
<!ELEMENT  NoteList         - - (Title?, (LI | LIBlock | Bridge)+) >
<!ATTLIST  NoteList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "SinglePartList"
    InfoMasterSemanticProperty
                     NAME         #FIXED    "Enumeration"
>
```

Notloc (notation-specific location)

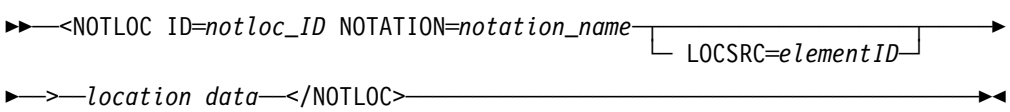
Purpose

Notloc is a HyTime element that contains a location specification unique to particular notation. For example, it could contain x and y pixel locations within a bitmap or the label of a graphic object in a CAD drawing. Notloc enables the definition of hyperlink anchors in non-SGML objects such as images. Notloc can only be refered to from a LinkDesc or RefKey element.

Valid Contexts

LinkDesc, RefKey.

Syntax



Attributes and Contained Elements

ID=*notloc_ID*
The ID of this Notloc element.

NOTATION=*notation_name*
The name of the notation that the specification is in. This does not have to be a declared SGML notation, although it can be.

LOCSRC=*elementID*
A reference to the element that contains or refers to the data being located into. To refer to a data entity, create a Nameloc element that refers to the entity using a NMList element with the ENTITY attribute specified.

location data
The notation-specific location specification. IBM ID Doc places no restrictions on the content of Notloc except that, as for all character data in an SGML document, it must be in the document character set. It is up to the system that provides link resolution services to properly handle the location specification.

Usage

Use Notloc to define hyperlink anchors in non-SGML objects such as images and vector graphics, or using query specifications in a specific notation. A hyperlink anchor can be the start of a link or the target of a link, or both. How a Notloc-defined anchor is used and expressed is a function of the online presentation system you are using and is not defined by IBM ID Doc.

Examples

In this example, the notation used is IPF's ARTLINK notation for defining link anchors in bitmap images.

```
<notloc id=ipfbitmap1 locsrc=mybitmap notation=ipfbitmap>
:artlink x=2 y=100.
</notloc>
```

SGML Definition

```
<!ELEMENT  NotLoc           - - (#PCDATA) -- HyTime notation location -->
<!ATTLIST  NotLoc
    ID              NAME         #REQUIRED
    Notation        NAME         #REQUIRED
    LocSrc          NAME         #REQUIRED

    -- HyTime attributes: --
    HyTime          NAME         #FIXED    "notloc"
    qdomain         NAMES        #IMPLIED
    qcontext        NAME         #IMPLIED
    ordering        (ordered)    ordered
              -- Up to presentation system to define meaning of order --
    set             (set)        set                 -- Ignore dupes --
    aggloc          (nagg)       nagg               -- Not aggregate --
>
```

Number

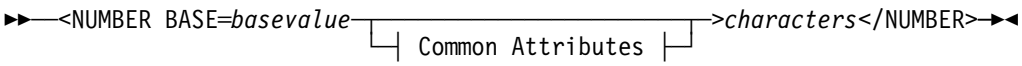
Purpose

The Number element identifies a number in some base other than binary, decimal, hexidecimal, or octal. Number is valid wherever PhraseLevelElements is valid.

Valid Contexts

PhraseLevelElements.

Syntax



Attributes and Contained Elements

BASE=*basevalue*
An integer value defining the base the number is in.

Usage

Use the Number element to identify numbers in a base for which there is not a more precise element, such as Hex or Bin. You must specify the Base= attribute to indicate the base of the number, for example, '36' for base 36 numbers.

Examples

```
<p>Numbers in base 34 use the digits zero (0) to nine (9)
and the letters A to Z minus I and O (or L and O), for
example, <number base=34>Z</number> = <decimal>33</decimal>.
```

SGML Definition

```
<!ELEMENT  Number           - - (#PCDATA) >
<!ATTLIST  Number
    Base            NUMBER       #REQUIRED
    %Common.Atts;

    InfoMaster      NAME         #FIXED    "Phrase"
>
```

ObjectContainer (data object container)

Purpose

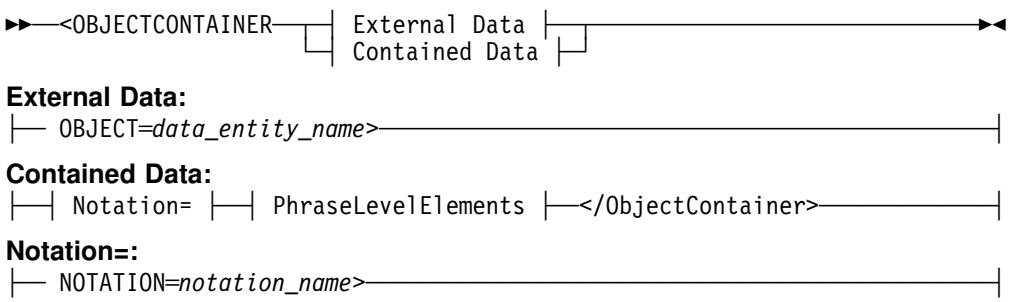
ObjectContainer contains or refers to a data object in a particular notation, such as a graphic, video clip, sound clip, spread sheet, program, and so on. ObjectContainer is normally used within the context of a MultiMediaObject, but may also be used in other contexts, such as within a figure.

Object container can contain a NotLoc element in order to define one or more hyperlink anchors within the object.

Valid Contexts

MultiMediaObject.

Syntax



Attributes and Contained Elements

OBJECT=*data_entity_name*
Refers to an external data entity that contains the object data. If you specify Object=, it is an error to specify any content or the ObjectContainer end tag.

NOTATION=*notation_name*
When the ObjectContainer data is specified inline, the Notation= attribute defines the notation the data is in. The supported notations are defined by the processing system.

contained data
The data comprising the object, for example, TIFF data.

Usage

The typical use of ObjectContainer is to contain or refer to graphics. ObjectContainer can be used to refer to any type of data that your processing system or presentation system supports. IBM ID Doc does not define or constrain the types of data you can use. However, most systems will support most of the typical graphics formats such as TIFF, CGM, EPS, and so on.

ObjectContainer can also be used to contain character data that is in a special notation, such as character graphics or a spread sheet saved as a text file.

When you use the OBJECT= attribute to refer to an external entity, you cannot specify the ObjectContainer end tag.

Note: You can define hyperlink points within an Object using the NotLoc= data attribute on the entity declaration for the data entity.

Examples

```
<MultiMediaObject>
<ObjectContainer object=GraphicEntityName>
<textalt>
<p>This describes the graphic with words.
</textalt>
</MultiMediaObject>
```

The next example demonstrates the use of the NotLoc= data attribute to define hyperlink points within the object.

```
<!DOCTYPE IBMIDDOC [
  ⋮
<!ENTITY graphic-one  SYSTEM "graphic1.tif" NDATA tiff
  [ notloc="x=100 y=400" ] >
  ⋮
]>
  ⋮
<MultiMediaObject>
 <ObjectContainer object=graphic-one>
 <TextAlt>
This graphic shows a....
</MultiMediaObject>
```

SGML Definition

```
<!-- These notations may be included in line within an SGML document.
     Other data which may not be included in line are referenced
     with a data entity reference, which identifies its notation
     on the entity declaration, not on the container element.
     Care must be taken when including graphics data
     inline to avoid misinterpretation of graphics commands as
     SGML markup. -->
<!ENTITY % ObjectContainer.Notations
    "EPS | CGMChar | %Math.Notations; |
     %ComputerLanguage.Notations; | %TextLanguage.Notations; |
     LineSpecific | LiteralData"
>

<!ELEMENT  ObjectContainer  - O (%DataPool;)*
                                              -- data object container --
>
<!ATTLIST  ObjectContainer
    Object           ENTITY       #CONREF
    Notation         (%ObjectContainer.Notations;)
                                  LiteralData
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    ApplicationClass NAME         #FIXED    "NonTextObject"
>
```

ObjectLibrary

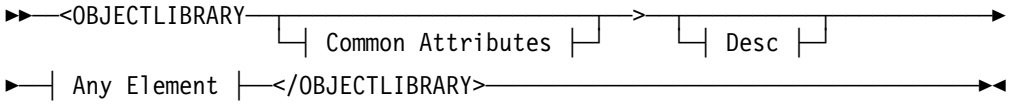
Purpose

ObjectLibrary contains elements that will be used elsewhere in the document by reference. ObjectLibrary is valid within DocProlog and DivProlog. You can have as many different ObjectLibrary elelments as desired. In addition, when cross-document object reference is supported, object libraries can be separate documents.

Valid Contexts

DivProlog, Prolog.

Syntax



Attributes and Contained Elements

Desc
A standard description element containing a description of this object library.

Any Elements
Any elements valid in the current document type.

Usage

Use object libraries to collect elements that you will re-use in many places. Elements within object libraries are not processed at the place they occur in the source document, but are only processed when refered to by another element of the same type using the common ContentRef= attribute or as the result of hyperlinks to the elements.

The object library can contain an optional description element intended to let you describe the purpose of the library.

Within the context of an object library, annotation elements (ANNOT, FN, etc.) are not implicitly associated with the object library itself, but are only associated with those elements that refer to them explicitly. This lets you use object libraries to organize elements such as footnotes in whatever way you want to without worrying about unexpected and unintended results.

Support for object libraries is optional.

Migration Note

ObjectLibrary replaces the function provided by DVCF side files for organizing arbitrary collections of text for re-use.

Examples

```
<ObjectLibrary>
<desc>This library contains definition list entries for user interface
elements that will be re-used in a number of places.
<dlentry id=filemenuitem class=menuitem>
<dt>File
<dd>Work with files.
</dlentry>
<dlentry id=Editmenuitem class=menuitem>
<dt>Edit
<dd>Perform edit functions.
```

```
</dlentry>
</ObjectLibrary>
```

SGML Definition

```
<!ELEMENT  ObjectLibrary    - O (Desc?, ObjectLibraryBody) >
<!ATTLIST  ObjectLibrary
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  ObjectLibraryBody
                              O O ANY
>
<!ATTLIST  ObjectLibraryBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Container"
>
```

Octal (octal number)

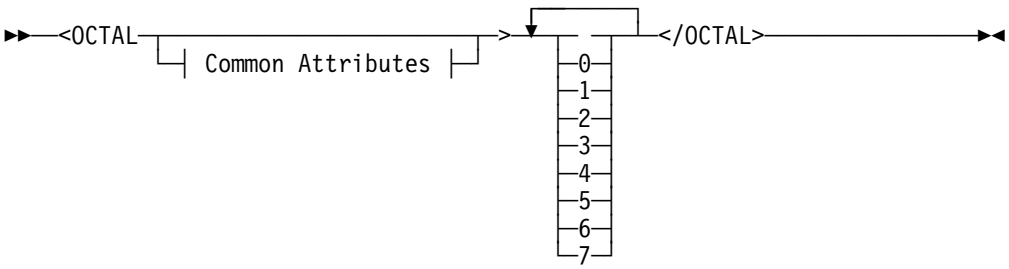
Purpose

The Octal element identifies octal data.

Valid Contexts

PhraseLevelElements.

Syntax



Usage

Use the Octal element to identify octal data. Octal data is data that is encoded in a base 8 numbering system. Its content is the eight octal digits, '0' thru '7', or space characters. A space character in the content of this number represents a visual separator within a single octal number (not a separator between multiple numbers).

Examples

```
<bin>11000001</bin> = <decimal>193</decimal> = <octal>301</octal>
```

SGML Definition

```
<!ELEMENT  Octal            - - (#PCDATA)        -- Octal data phrase -->
<!ATTLIST  Octal
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "OctalData"
    LexModel         CDATA        #FIXED
                          "('0'|'1'|'2'|'3'|'4'|'5'|'6'|'7' | ' ')*
                           &Common.LexModel;"
>
```

OL (ordered list)

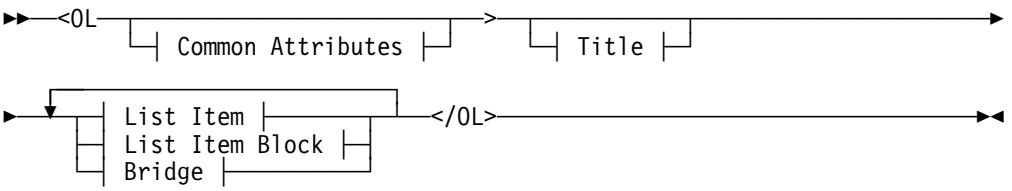
Purpose

Contains a single part list in which the items' order is important.

Valid Contexts

ParaLevelElements

Syntax



Attributes and Contained Elements

Title
Use Title to define a title for the list.

List Item
A list item.

List Item Block
Organizes list items within the list.

Bridge
A Bridge element, normally used to define a transition or linkage between two list item blocks.

Usage

Ordered lists are valid anywhere paragraphs are valid. Paragraphs can contain lists, so you must explicitly end a paragraph before a list if you want the list to be outside the paragraph.

Examples

Simple:

```
<ol>
<li>bread
<li>milk
<li>peanut butter
</ol>
```

More complex:

```
<ol>
<li>bread:
<ol compact>
<li>whole-wheat flour
<li>sandwich-sized slices
</ol>
<li>milk
<li>peanut butter
<ol compact prefix='checkoff setup' setuptext='Ingredient'>
<li>peanuts
<li>salt
<li>dash of sugar
</ol>
</ol>
```

SGML Definition

```
<!ELEMENT  OL              - - (Title?, (LI | LIBlock | Bridge)+)
                                                    -- Ordered List --
>
<!ATTLIST  OL
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "SinglePartList"
    InfoMasterSemanticProperty
                     NAME         #FIXED    "Enumeration"
>
```

Oper (syntactic operator)

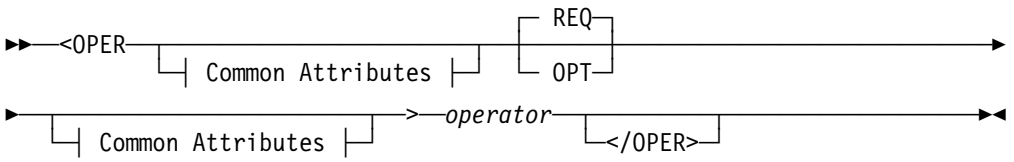
Purpose

Defines an operator within a syntax definition group, fragment, or SynPhrase.

Valid Contexts

Group, RepSep, SyntaxPhrase.

Syntax



Attributes and Contained Elements

REQ | OPT
Indicates whether or not the operator is optional. REQ (required) is the default.

operator
The operator characters.

Usage

Use Oper to define operators within a syntax definition or SynPhrase element. Operators are normally connectors that connect two parts of a statement and imply some action, such as assignment or comparison. Operators can also be unary operators that are applied to a single parameter, such as the negation operator. Typical operators are the equals sign (=) and the mathematical operators.

Examples

```
<syntax>Attribute Specification
  <group>Attspec
    <sep>&ssbl;
    <kwd>ATTNAME
    <oper>=
    <var>value
  </group>
</syntax>
```

SGML Definition

```
<!ELEMENT  Oper             - O (%DataPool;)*      -- Syntax Operator -->
<!ATTLIST  Oper
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;
>
```

P (paragraph)

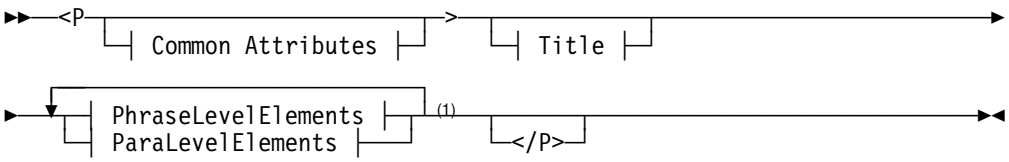
Purpose

Contains a block of text representing a single idea. Paragraphs can contain other elements, such as lists.

Valid Contexts

ParaLevelElements.

Syntax



Note:

[1] Paragraphs cannot directly contain paragraphs (but they can contain elements that contain paragraphs, such as lists).

Attributes and Contained Elements

Migration Note

Paragraph titles replace the use of low-level headings like H5 and H6 out of the normal heading sequence. Paragraph titles also replace constructs such as:

```
:p.:hp2.Examples::ehp2.
:p....
```

ParaLevelElements
PhraseLevelElements

Paragraphs can contain almost anything except divisions. Paragraphs cannot directly contain other paragraphs.

Migration Note

Note that the need for the paragraph continuation element (:PC) goes away because the paragraph element now has an end tag. Text following an element contained within a paragraph is always part of the paragraph and does not need further markup.

Usage

Use paragraphs to contain flowing text and other elements associated with the text, such as lists, examples, figures, and tables.

Use the paragraph end tag to control whether or not other elements are contained within the paragraph. For example, in the following example, the unordered list is contained within the paragraph:

```
<p>This is a list:
<ul>
<li>List item
</ul>
```

In order to keep the list from being part of the paragraph, you must explicitly end the paragraph before the list:

```
<p>This is separate from the list.
</p>
<ul>
<li>List item
</ul>
```

Paragraphs cannot themselves directly contain other paragraphs, so you do not need to end a paragraph before starting another paragraph.

Paragraphs with no content are not valid.

Examples

Simple example:

```
<p>paragraph content
```

Example of bounding:

```
<p>text paragraph content
<ul>
<li>list item content within paragraph
</ul>
more text paragraph content
</p>
```

SGML Definition

```
<!ELEMENT  P               - O (Title?, ParaBody)       -- Paragraph -->
<!ATTLIST  p
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
>
<!ELEMENT  ParaBody         O O (%InfoPool.NoPara;)*
                                                -- paragraph body --
>
<!ATTLIST  ParaBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ParagraphBody"
>
```

ParaBlock (paragraph block)

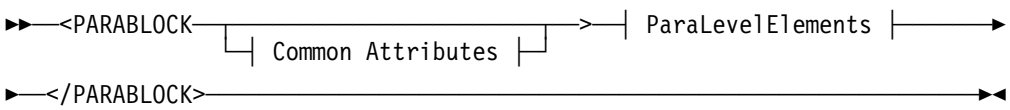
Purpose

Parablock lets you group paragraphs and paragraph-level elements, such as lists and figures, into logical groupings, for example, to assign property values to the contained elements or to apply style specifications to the contained elements.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

ParaLevelElements

Parablock can contain anything directly containable by divisions except divisions.

Usage

Use ParaBlock to group paragraph and paragraph-level elements together arbitrarily. The default style for ParaBlock is to keep elements within the block together on a page. Parablock can be used to define property values for a set of elements. For example, if a number of paragraphs have changed, you can put them within a ParaBlock element in order to define their revision status.

You can also use ParaBlock within an ObjectLibrary to define groups of paragraphs for use by reference. For example, you may have a figure with an introductory paragraph that you would like to re-use in several contexts. You can put the paragraph and the figure in a ParaBlock and then use a ParaBlock with a ContentRef= attribute to use the contents of the ParaBlock in multiple places.

ParaBlock also gives you a way to define a set of paragraphs as a single hyperlink anchor by linking to the ParaBlock element.

Examples

```
<ParaBlock>
<p>This is a paragraph.
</p>
<ul>
<li>Part of the block
<li>Part of the block
</ul>
<p>Another paragraph in the parablock.
</ParaBlock>
```

SGML Definition

```
<!ELEMENT  ParaBlock        - - (Title?, (%InfoUnitPool;))*
                                                -- Paragraph block --
>
<!ATTLIST  ParaBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

ParaLevelElements (paragraph-level elements)

Purpose

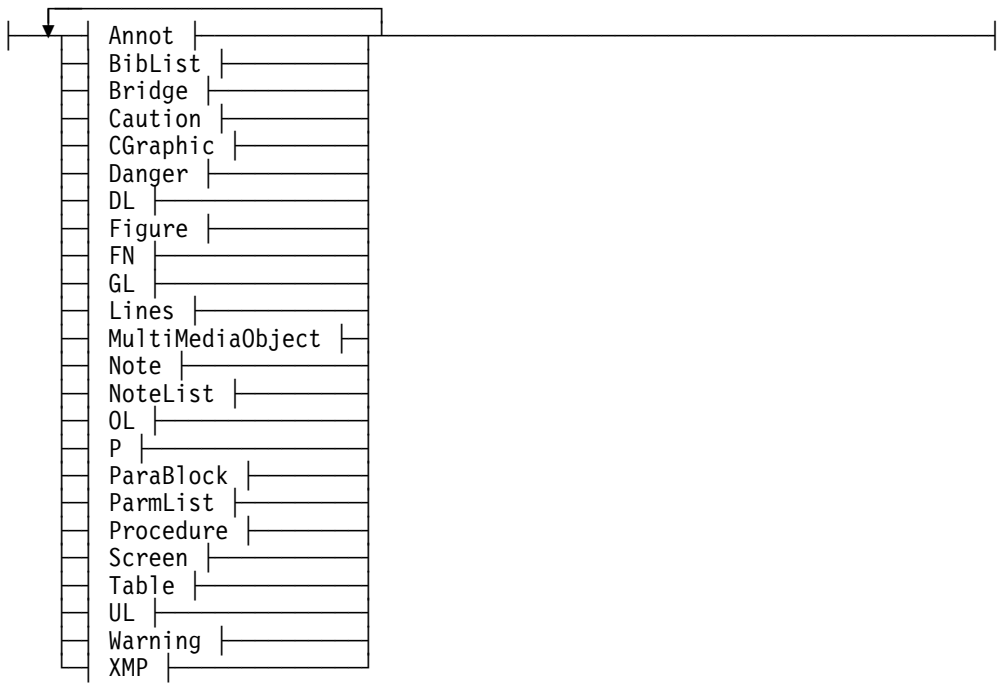
The ParaLevelElements elements are the paragraph-level elements, essentially the directly-containable content of divisions, except for divisions themselves. ParaLevelElements' content includes some phrase-level elements and may also contain other paragraph-level elements.

Valid Contexts

Most elements that are not limited to phrase-level element content.

Syntax

ParaLevelElements:



Usage

See the descriptions of the various elements for details on using those elements.

SGML Definition

```
<!--InfoUnitPool is the paragraph-level elements, essentially, division
    content.  Built from more detailed parameter entities.
-->

<!-- This entity contains those elements which are common between
     InfoUnitPool and DataPool.  The elements on the first line of the
     definition are from the base, the following lines each contain
     elements from a separate tower.
-->
<!ENTITY % InfoUnitPool.DataPool.Common
```

ParaLevelElements (paragraph-level elements)

```
    "Link | nameloc | XRef |
     MultiMediaObject |
     FN | Citation |
     %BibRecord; |
     IRef | I1 | I2 | I3 |
     Formula |
     RefKey"
>

<!-- The following entity contains all the InfoUnitPool elements that
     are defined as towers.  Each line contains a separate tower.
-->
<!ENTITY % InfoUnitPool.Towers
    "Annot |
     BibList | FigureList | TableList | FootnoteList |
     CGraphic |
     GL |
     %LibRecord; |
     LERS |
     Lines |
     LitData |
     LongQuote |
     ManPage |
     MarkedNote | MarkList |
     MessageList |
     NoteList |
     ParmList |
     Procedure |
     RetrievalAlternatives |
     Screen | XMP |
     Syntax"
>

<!-- The following entity contains the base InfoUnitPool elements
     except P, Note and ParaBlock since these are included below.
     As currently setup, P, Note and ParaBlock are all part of the base.
-->
<!ENTITY % InfoUnitPool.Base
    "Bridge |
     DL | OL | UL |
     Figure | InfoModule |
     ModularInformation |
     Table"
>

<!-- The following 3 entities are used in content models of the DTD. -->
<!ENTITY % InfoUnitPool.NoPara
    "Note | Caution | Danger | Warning |
     (%InfoUnitPool.Base;) |
     %InfoUnitPool.DataPool.Common |
     %InfoUnitPool.Towers;"
>

<!ENTITY % InfoUnitPool.NoNote
    "P | ParaBlock |
     (%InfoUnitPool.Base;) |
     %InfoUnitPool.DataPool.Common |
```

```
     %InfoUnitPool.Towers;"
>

<!ENTITY % InfoUnitPool
    "P | ParaBlock | Note | Caution | Danger | Warning |
     (%InfoUnitPool.Base;) |
     %InfoUnitPool.DataPool.Common |
     %InfoUnitPool.Towers;"
>
```

Parm (parameter list entry)

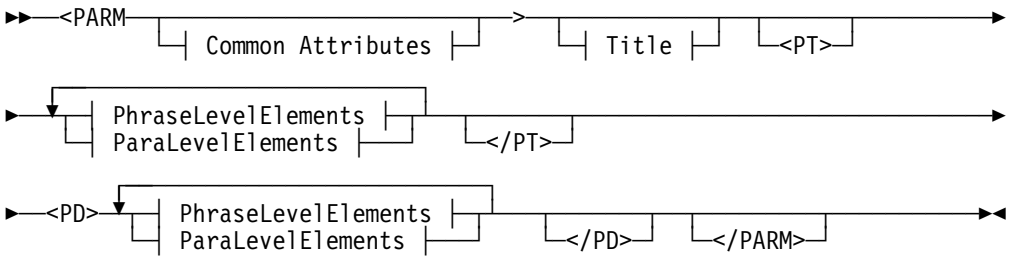
Purpose

Contains a single parameter and its definition within a parameter list.

Valid Contexts

ParmList, ParmBlock.

Syntax



Attributes and Contained Elements

Title
The parameter list can have a title.

PT Contains the parameter itself. PT can contain anything that is valid within phrase- or paragraph-level elements. The PT tag is omissible.

When parameter lists are used with syntax definitions, the typical content for PT is a SynPhrase element containing the definition of the syntax of the parameter being described.

PD
Contains the description of the parameter.

Usage

Use Parm to describe a single parameter in some sort of computer language or similar construct.

Examples

```
<ParmList>
<parm><synphrase><kwd>ATTNAME<oper>=<var>value</synphrase>
<pd>All attribute specifications consist of an attribute
name, the assignment operator (<synprhase><oper>=</synphrase>),
and a value.
</ParmList>
```

SGML Definition

```
<!ELEMENT  Parm            - O (PT, PD)
                                -- Parmeter Definition List Entry --
>
<!ATTLIST  Parm
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Term"
>

<!ELEMENT  PT              O O (%InfoPool;)*  -- Parameter List Term -->
<!ATTLIST  PT
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinedTerm"
>

<!ELEMENT  PD              - O (%InfoPool;)*
                                -- Parameter List Term Definition --
>
<!ATTLIST  PD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermDefinition"
>
```

ParmBlock (parameter list block)

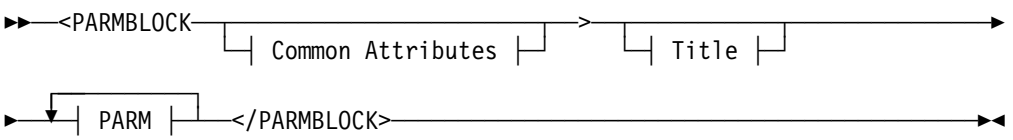
Purpose

Organizes parameter list entries within a parameter list.

Valid Contexts

Parmlist.

Syntax



Attributes and Contained Elements

Title
An optional title for the parameter block.

PARM
Contains a single parameter description.

Usage

ParmBlock has an optional title. You can use ParmBlock to organize parameters within a list into meaningful groupings. For example, within the definition of a complex statement or command, you may have grouped parameters logically. You could use ParmBlock within a single parameter list to mirror the grouping in the syntax definition.

Examples

```
<ParmList>
<ParmBlock><title>Option Group 1
<parm>FOO=bar
<pd>
<parm>BAR=foo
<pd>
</parmblock>
<ParmBlock><title>Option Group 2
<parm>A=B
<pd>
<parm>B=C
<pd>
<parm>A=C
<pd>
</ParmBlock>
</ParmList>
```

SGML Definition

```
<!ELEMENT  ParmBlock        - O (Title?, (Parm | Bridge)+)
                                    -- Parameter List Block of Entries --
>
<!ATTLIST  ParmBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionBlock"
>
```

ParmList (parameter list)

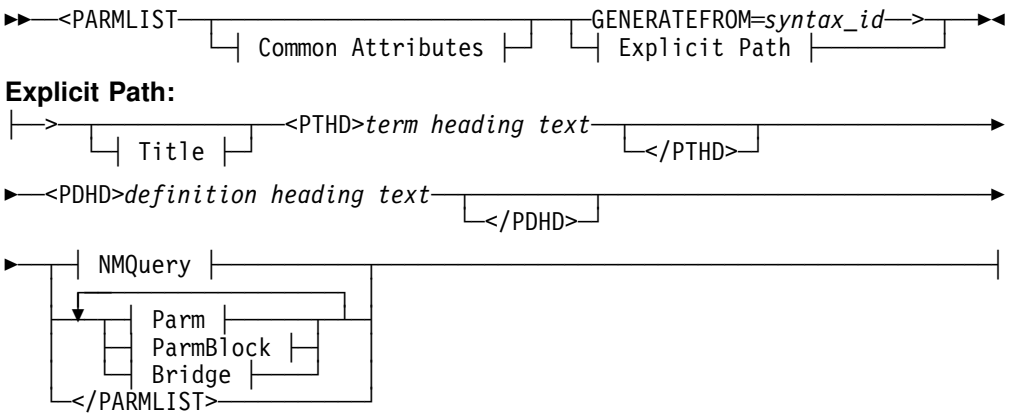
Purpose

Contains computer language parameter and definition pairs, possibly organized by ParmBlock elements.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

GenerateFrom=*syntax_id*

Refers to a syntax definition from which to generate the content of the parameter list. Each Group within the syntax definition that contains a Desc element will cause the generation of a parameter entry where the parameter term is the syntactic content of the Group and the parameter description is the content of the Desc element.

When the GenerateFrom= attribute is specified, it is an error to specify any other ParmList elements or the ParmList end tag.

The GenerateFrom= attribute causes the default ParmList query to be used to generate the contents of the parameter list. You can also specify an explicit query by using the a NMQuery element as the content of the ParmList element.

Title

The parameter list can have a title.

PTHD

Defines the heading text for the parameter terms. In the default presentation style, the PTHD is used as a column heading.

PDHD

Defines the heading text for the parameter definitions. In the default presentation style, the PDHD is used as a column heading.

NMQuery

Defines the query to be used to generate the content of the parameter list. For example, you could use NMQuery within parameter list to retrieve a list of all parameters defined within a document, not just within a single syntax definition.

Parm
ParmBlock
Bridge
The Parm, ParmBlock, and Bridge elements can be used together within ParmList in any combination. The typical use of Bridge is to provide a transition between two groups of parameters organized within ParmBlock elements.

Usage

Use parameter lists to contain the description of parameters in a computer language statement. Parameter lists are normally associated with syntax definitions. Entries can be organized within parameter lists using parameter block (ParmBlock) elements. Bridge elements can also be used to create transitions or connections between blocks of entries. Parameter lists have optional titles.

The parameter element can contain any ParaLevelElements elements, including syntax definitions.

Examples

```
<parmlist>
<parm>
&parmsyn; <!-- Reference to syntax markup -->
<pd>Definition
</parmlist>
```

SGML Definition

```
<!ELEMENT  ParmList         - O (Title?, (PTHD, PDHD)?,
                                 (nmquery |
                                  (Parm | ParmBlock | Bridge)+
                                ))
                                         -- Parameter Definition List --
>
<!ATTLIST  ParmList
    GenerateFrom     NAME         #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefiningList"
>
```

Part (major document part)

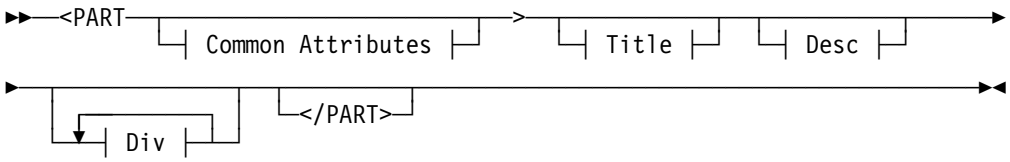
Purpose

Organizes the document body into major logical parts, which function as a sort of mini document within a document. The Part element does not change the logical hierarchy of the divisions it contains. In other words, if in your document style, the first level of division is considered to be a chapter, they will still be chapters when contained within Part. In the default style, the enumeration of divisions contained within Parts is not affected by the presence or absense of Part elements.

Valid Contexts

Body, Div.

Syntax



Attributes and Contained Elements

Title
The part division title.

Desc
A description of the part.

Div
Part can contain only divisions.

Usage

Use Part to divide a document into logical sub-documents. For example, in a document that contains both usage and reference information you might define two parts, one containing the usage information and one containing the reference information. A Part can have a title and a description. The description is intended to contain a description of the part's purpose and introduce its content.

In the default presentation style, Parts do not affect the enumeration of contained divisions. For example, if in your document's style, the first level divisions are numbered and labeled as chapters, the presence or absense of Parts will not affect the numbering. In other words, the numbering of the chapters would not be reset in a new part.

Part has an optional Title.

Examples

```
<Part>Using &prodname;
<div>First Division
</div>
<div>Second Division
</div>
<Part>&prodname; Reference
```

SGML Definition

```
<!ELEMENT  Part             - O (Title?, Desc?, Div*) >
<!ATTLIST  Part
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

Person (person's name and address)

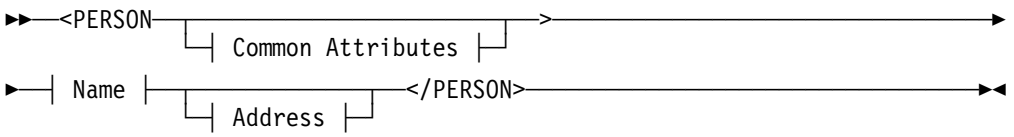
Purpose

Contains the name and address of a person, as opposed to an enterprise.

Valid Contexts

Author, Approvers, Owners.

Syntax



Attributes and Contained Elements

Name
Contains a name.

Address
Contains an address.

Usage

The elements Person and Enterprise both contain name and address pairs for use in contexts like Author, Approvers, and Owners where either a person or an enterprise could be meaningful. The main distinction between Person and Enterprise is that Enterprise has a special name element, EnterpriseName that has different subelements than Name does.

Examples

```
<author role="Editor">
 <person>
  <name id=drmacro><last>Kimber<first>W<middle>Eliot<honorific>Mr.</name>
  <address>
  ⋮
  </address>
 </person>
</author>
```

SGML Definition

```
<!ELEMENT  Person           - O (Name, Address?) >
<!ATTLIST  Person
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "NameAndAddress"
>
```

Phrase (one or more words)

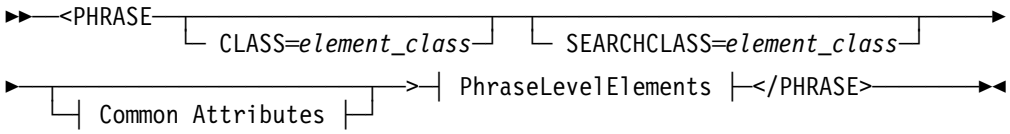
Purpose

The Phrase element allows you to identify a phrase (one or more words) for some purpose. Phrase can either be used to define containment structures for the purpose of associating one element with another, such as associating a footnote with a specific sentence; or to further specify the semantic meaning of a phrase by using an author-defined element class.

Valid Contexts

PhraseLevelElements.

Syntax



Attributes and Contained Elements

CLASS=*element_class*

The *phrase_type* is an author-defined class to be associated with the phrase. Element classes are defined using the ClassDef element. Any element class that is not defined as being restricted to a particular element type can be used as the phrase class.

SEARCHCLASS=*element_class*

Defines the class of element that the phrase should be linked to for the purpose of resolving associative links. When SearchClass= is not specified, the default hyperlink behavior is for phrases to be associated with other elements of the same class.

PhraseLevelElements

Phrase can contain any PhraseLevelElements element, including Phrase.

Usage

Use the Phrase element to identify a phrase for some reason not already provided for by the IBM ID Doc language. For example, you may want to associate a revision level, version level or other processing semantic with a couple of words within a paragraph. Or you may want to identify a word as a particular type of data for special processing.

You can provide very precise identification of information that is unique to your document by defining element classes with the ClassDef element and then using those classes with the Phrase element. For example, in the documentation for a program that supports mining operations, you may need to talk about different kinds of rocks and would like to precisely identify references to different rocks to enhance the retrievability of your information. You could define element classes for the different

rock types and then use Phrase elements with those classes to identify references to types of rock.

There are several applications of phrase elements that are very useful:

- associative linking

  IBM ID Doc enables associative linking based on author-defined classes of phrases. In practice, this means that if special classes of data are identified, the content of this data can be linked automatically to any occurrences of this data throughout the text. Element classes can also be used to limit the scope of associative linking so that only special uses of given phrases become targets for associative links.

- automatic indexing

  Automatic indexing and associative linking are somewhat related. Defining the specialized context for terms allows indexing software to be more selective in the entries that it includes in a generated index. Identified properly, it also allows the software to attach appropriate levels of importance to these entries.

  You can define index entry text to be associated with different element classes defined with ClassDef. For any phrase with a class specified, an index entry can be generated consisting of the phrase content itself and the entries as defined by the class hierarchy of the phrase's class. See “ClassDef (element class definition)” on page 74 for more information on defining element classes and automatic generation of indexes.

- search context

  Just as associative linking and automatic indexing can use the context to limit their searching, searches and queries can use phrase classes to limit the scope of searches. Because the element classes can precisely reflect the content of a given document, searches can be made very accurate and precise, optimizing retrieval of information on a specific topic.

Examples

To indicate material has changed from one revision to another:

```
&prod; is supported on
<phrase rev=rel1>VM.</phrase>
<phrase rev=rel2>VM and MVS.</phrase>
<phrase rev=rel3>OS/2, VM and MVS.</phrase>
```

To mark a particular class of phrase (in this case for NLS considerations):

```
<prolog>
<PropertyDefinitions>
  ⋮
<ClassDef classname=cliche>cliches
 <semantic>Identifies phrases that are cliches in the
 current language and that may need special attention
 during translation.
</ClassDef>
</PropertyDefinitions>
</prolog>
  ⋮
```

```
<p>
&prodname;is <phrase class=cliche>the greatest thing since sliced
bread</phrase>.
```

SGML Definition

```
<!ELEMENT  Phrase           - - (%DataPool;)*       -- Generic phrase -->
<!ATTLIST  Phrase
    SearchClass      NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
>
```

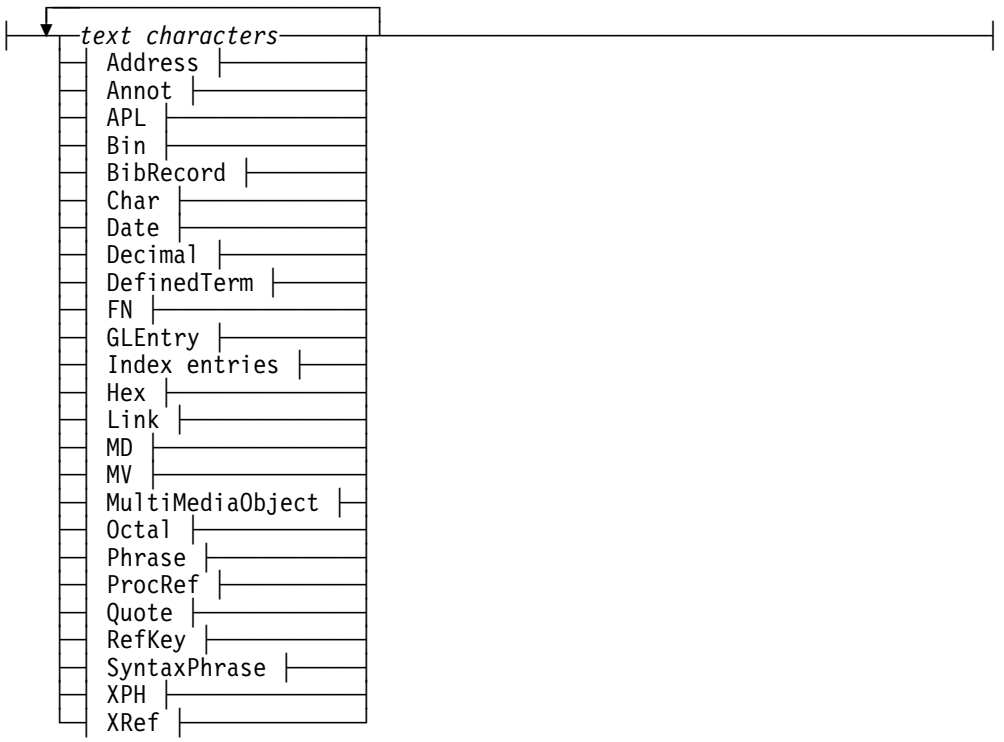
PhraseLevelElements

Purpose

PhraseLevelElements represents all the phrase-level elements. Except for within elements with very specific content rules, any PhraseLevelElements element is valid anywhere plain text is valid.

Syntax

PhraseLevelElements:



Attributes and Contained Elements

See the description of each element for details on using that element.

Usage

The PhraseLevelElements elements are valid anywhere text is valid except for within elements with very precise content restrictions, such as Phone or Zip.

SGML Definition

```
<!--DataPool defines the phrase-level elements.  Elements that are
    either DataPool or InfoUnitPool are defined separately.
-->
<!--DataPool defines the phrase-level elements.  Elements that are
    either DataPool or InfoUnitPool are defined separately.
-->

<!ENTITY % DataPool.Towers
    "Address |
     APL |
     Quote |
     Bin | Char | Decimal | Hex | Number | Octal |
     Date |
     GLEntry |
     MD |
     MV |
     PK | PV |
     SyntaxPhrase |
     ProcRef |
     XPH"
>
<!ENTITY % DataPool.Base    "#PCDATA | Phrase | DefinedTerm">

<!ENTITY % DataPool
    "%DataPool.Base; |
     %DataPool.Towers; |
     %InfoUnitPool.DataPool.Common;"
>
```

Preface

Purpose

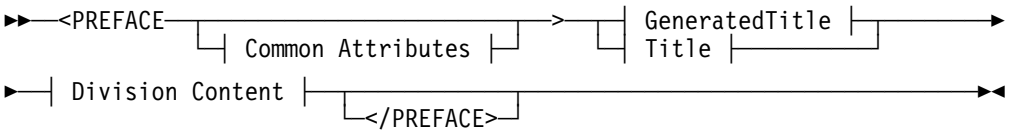
Preface is intended to contain preferatory information, such as how to use the information, the purpose of the information, and other ancillary information.

Valid Contexts

Frontmatter.

Procedure

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the Preface division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

Division Content

Preface has the same content rules as Div elements.

Usage

Examples

```
<frontmatter>
<preface><title>About This Information</title>
<p>This information is...
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  Preface          - O (%SpecialDiv;) >
<!ATTLIST  Preface
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

Procedure

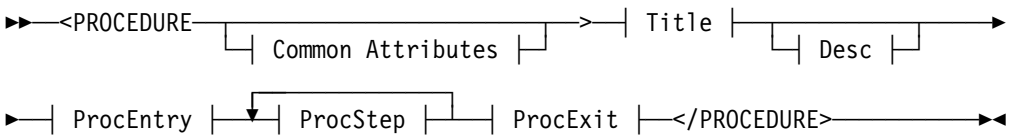
Purpose

The Procudure element structures information that describes a procedure or task. The intent of the Procedure element is to include all the information needed to completely define a procedure to the degree that the procedure markup could drive automated processes or generate computer-based training, for example. The markup also enables a wide variety of print and online presentation styles. Finally, it contains specific hyperlink elements for connecting procedure definitions together and for connecting procedure descriptions to graphics and other multimedia elements, such as animation or tutorials related to the procedure.

Valid Contexts

Body, ParaLevelElements.

Syntax



Attributes and Contained Elements

Title
The title of the procedure. The title start and end tags can be omitted. Title is required.

Desc
A description of the procedure as a whole.

ProcEntry
Describes the entry point to the procedure.

ProcExit
Describes what to do at the end of the procedure, such as where to go next or what the expected results are.

ProcStep
Describes one step of the procedure.

Usage

Use the procedure element to describe procedures such as user tasks. A procedure consists of three basic parts, a procedure entry, one or more steps, and a procedure exit. The procedure entry defines the entry criteria for a procedure, such as any pre-requisite tasks and related tasks. Each step then defines the actions to take and the expected results. The procedure exit describes the expected result of performing the task and where to go next.

A step can itself contain one or more procedures within it.

Examples

```
<procedure id=outerproc>Procedure X
  <Desc>
  <p>Use this procedure to do X.
 <ProcEntry>
  <!-- The next two elements contain IDs of other procedures -->
  <Prerequisites>proc0
  <RelatedProcedures>proc1 proc2 proc3
 </ProcEntry>
 <ProcStep id=procstepA>
   <title>First thing you do
   <Desc>
   <p>This step contains a nested procedure:
   <Procedure id=nestedproc>
     <ProcEntry>
```

```
      <Prerequisites>procA
      <RelatedProcedures>procB procC
      <Desc>
      <p>This is a subprocedure within the first step of "outerproc".
     </ProcEntry>
     <ProcStep id=nestedprocA>
       <title>First step of nested proc
       <Desc><p>Do something.
  </procedure>
  <StepNotes>
    <Caution>Perform this step at your peril.
 <ProcStep id=procstepB>
  <title>Second thing you do
  <!-- ReferenceKey, below, lists the IDs of Reference Key elements (RK).
     Reference Keys define points in graphics or text display elements
     (e.g., XMP, SCREEN).  Reference Keys thus defines a linkage from
     this step to specific points in an explanatory graphic.  The
     Reference Key element itself could also define a link from the
     graphic to one or more procedure steps.
    -->
  <ReferenceKeys>rk1 rk6 rk7
  <Desc><p>Do even more stuff.
  <DecisionPoint>
    <Condition>You are done
    <Action>Go to step <StepRef refid=procexita>.
    <Condition>You are not done
    <Action>Go to step <StepRef refid=procstepA>.
  </DecisionPoint>
 <ProcExit id=procexita>
   <ExpectedResults><p>What should have happened.
   <RecoveryProcedure><p>What to do if things didn't go well.
   <Descn><p>Talk about the end of the procedure.
 </ProcExit>
</Procedure>
```

SGML Definition

```
<!ELEMENT  Procedure        - -
                     (Title, Desc?, ProcEntry, ProcStep+, ProcExit)
>
<!ATTLIST  Procedure
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
    ApplicationClass NAME         #FIXED    "Procedure"
>
```

ProcEntry (procedure entry point)

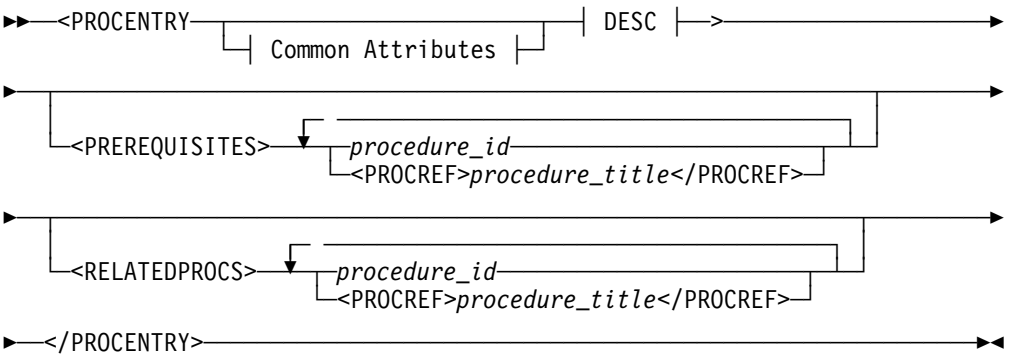
Purpose

The ProcEntry element defines the entry criteria for a given procedure and describes the procedure itself.

Valid Contexts

Procedure.

Syntax



Attributes and Contained Elements

Desc
Contains a description of the procedure entry conditions.

Prerequisites
Contains a list of procedure IDs refering to prerequisite procedures or LinkDesc elements (for reference to procedures in other documents) or ProcRef elements. The order the IDs and ProcRefs are specified indicates the order the prerequisite procedures should be performed.

RelatedProcs
Contains a list of related, but not prerequisite, procedure IDs, LDESC elements, ProcRef elements,

ProcRef
Contains a logical, but not literal, reference to a procedure. ProcRef enables reference to a procedure that does not have a formal description.

Usage

Use ProcEntry to indicate any prerequisite procedures or related procedures. ProcEntry is required.

Examples

```
<procedure id=outerproc>Procedure X
 <ProcEntry>
  <!-- The next two elements contain IDs of other procedures -->
  <Prerequisites>proc0
  <RelatedProcs>proc1 proc2 proc3 <procref>Z proc</procref>
  <Desc>
  <p>Use this procedure to do X.
 </ProcEntry>
```

SGML Definition

```
<!ELEMENT  ProcEntry        - O (Desc, Prerequisites?, RelatedProcs?)
                                      -- Procedure entry information --
>
<!ATTLIST  ProcEntry
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ProcEntry"
>

<!ELEMENT  (Prerequisites | RelatedProcs)
                              - O (#PCDATA | ProcRef)*
>
<!ATTLIST  (Prerequisites | RelatedProcs)
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                          "#CONTENT (NAME | #GI)*
                           &Common.LexModel;"
    RefType          CDATA        #FIXED
                          "#CONTENT (Procedure | LinkDesc)
                           &Common.RefType;"
>
```

ProcExit (procedure exit point)

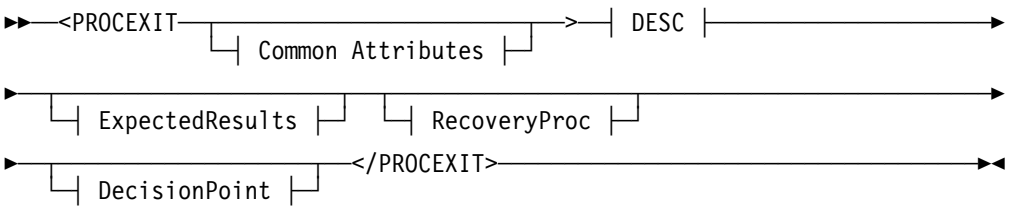
Purpose

ProcExit describes the exit criteria for a procedure and optionally contains information about what to do next and how to recover if something went wrong.

Valid Contexts

Procedure, following any ProcStep elements.

Syntax



Attributes and Contained Elements

Desc
The description of the end of the procedure.

ExpectedResults
What should have happened or what you should have as a result of completing the procedure.

RecoveryProc
What to do if things didn't go well. This can be a simple prose description of what to do, a complete procedure, or a reference to a procedure (ProcRef).

DecisionPoint
Identifies a decision point in a procedure. The decision point element contains one or more condition and action pairs. See “DecisionPoint” on page 92 for details..

Usage

Examples

```
<Procedure>
  ⋮
  <ProcExit id=procexita>
    <Desc><p>Talk about the end of the procedure.
    <ExpectedResults><p>What should have happened.
    <RecoveryProc><ProcRef>Recovery Procedure A
  </ProcExit>
</Procedure>
```

SGML Definition

```
<!ELEMENT  ProcExit         - O
                     (Desc, ExpectedResults?, RecoveryProc?,
                      DecisionPoint?)
                                      -- Procedure exit information --
>
<!ATTLIST  ProcExit
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ProcExit"
>

<!ELEMENT  (ExpectedResults | RecoveryProc)
                               - O (%InfoPool;)*
>
<!ATTLIST  (ExpectedResults | RecoveryProc)
    %Common.Atts;
>
```

ProcStep (procedure step)

Purpose

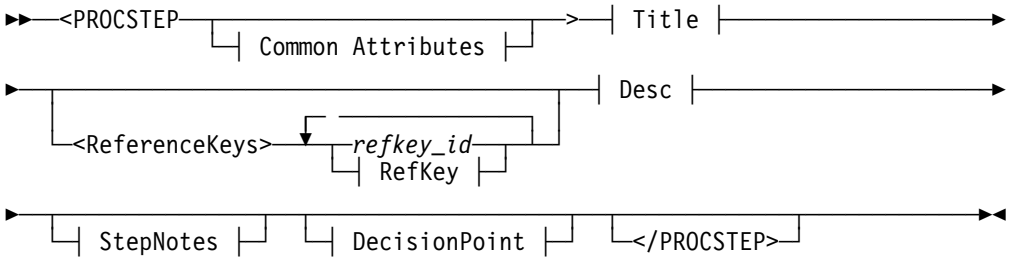
Describes a single step in a procedure.

ProcStep (procedure step)

Valid Contexts

Procedure, following ProcEntry.

Syntax



Attributes and Contained Elements

Title
The title of the step.

ReferenceKeys
Contains reference key elements (RefKey) or RefKey element IDs, creating a linkage between the step and whatever the reference key refers to (presumably a graphic or other non-text object).

ReferenceKeys is optional.

RefKey
A RefKey element.

refkey_id
The ID of a RefKey element.

Desc
The description of the step. The description can itself contain another procedure.

StepNotes
Contains one or more list item elements, each item containing an explanatory note about the step. StepNotes is a specialized version of NoteList.

DecisionPoint
Identifies a decision point in a procedure. The decision point element contains one or more condition and action pairs. See “DecisionPoint” on page 92 for details..

Usage

A procedure is made up of one or more procedure steps. Each step contains a description of the step followed by an optional decision point specification indicating what further action to take. The default action is to procede to the next step in the procedure. A step can also contain a StepNotes element containing notes about the step. StepNotes is a specialized version of NoteList.

A procedure step description can itself contain a procedure, allowing you to nest procedures to any level desired (within the general element nesting limits imposed by IBM ID Doc).

The ReferenceKeys element lets you associate a step with some explanatory information, such as a graphic or other non-text object. You can specify reference keys directly with the RefKey element or specify reference key IDs. See “RefKey (reference key hyperlink)” on page 281 for more information on using reference keys.

Examples

```
<ProcStep id=procstepA>
  <title>First thing you do
  <Desc>
  <p>This step contains a nested procedure:
  <Procedure id=nestedproc>
    <ProcEntry>
     <Prerequisites>procA
     <RelatedProcs>procB procC
     <Desc>
     <p>This is a subprocedure within the first step of "outerproc".
    </ProcEntry>
    <ProcStep id=nestedprocA>
      <title>First step of nested proc
      <Desc><p>Do something.
  </procedure>
  <StepNotes>
    <Caution>Perform this step at your peril.
<ProcStep id=procstepB>
  <title>Second thing you do
  <!-- ReferenceKeys, below, lists the IDs of Reference Key elements (RefKey).
     Reference Keys define points in graphics or text display elements
     (e.g., XMP, SCREEN).  Reference Keys thus defines a linkage from
     this step to specific points in an explanatory graphic.  The
     Reference Key element itself could also define a link from the
     graphic to one or more procedure steps.
    -->
  <ReferenceKeys>rk1 rk6 <refkey>C</>
  <Desc><p>Do even more stuff.
  <DecisionPoint>
    <Condition>You are done
    <Action>Go to <XRef refid=procexita>.
    <Condition>You are not done
    <Action>Go to <XRef refid=procstepA>.
  </DecisionPoint>
  ⋮
</Procedure>
```

SGML Definition

```
<!ELEMENT  ProcStep         - O
                      (Title, ReferenceKeys?, Desc, StepNotes?,
                       DecisionPoint?)
                                                  -- Procedure step --
>
<!ATTLIST  ProcStep
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ProcStep"
>
```

Prolog (document metainformation)

```
<!ELEMENT  ReferenceKeys    - O (#PCDATA | RefKey)* >
<!ATTLIST  ReferenceKeys
    %Common.Atts.NoLexMdl;

    LexModel         CDATA        #FIXED
                          "#CONTENT (NAME | #GI)
                           &Common.LexModel;"
>
```

Valid Contexts

Procedure, ProcEntry, ProcExit, RefKey, DecisionPoint

Prolog (document metainformation)

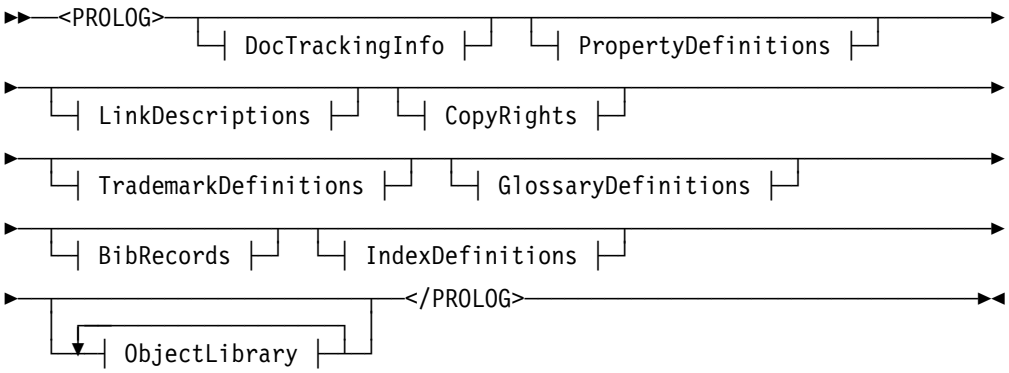
Purpose

Prolog contains all the descriptive metainformation about the document. Metainformation is that information that describes the document itself, such as DocTitle, Author, and DocumentNumber.

Valid Contexts

The first element within IBMIDDOC.

Syntax



Attributes and Contained Elements

DocTrackingInfo
Contains the tracking and control information for the document, such as document title, control numbers, revision definitions, applicable change and control document references, ownership, security classifications, and so on.

PropertyDefinitions
All property definitions applicable to the entire document.

LinkDescriptions
Contains link description elements (LinkDesc).

CopyRights
Defines copyright information for the document.

TrademarkDefinitions
Contains trademark definitions for use elsewhere in the document.

GlossaryDefinitions
Contains GLEntry elements for use by reference from anywhere in the document.

BibRecords
Contains BibRecord and LibRecord elements for documents and libraries that apply to the entire document.

IndexDefinitions
Contains index entries that will be used by reference from within the document.

ObjectLibrary
You can use object libraries to organize re-usable elements. Elements within object libraries will only be processed if they are refered with ContentRef= attributes or by other types of hyperlinks. ObjectLibrary can contain any element.

Usage

Use Prolog to contain all the descriptive information for a document.

Examples

```
<IBMIDDOC>
<prolog>
<DocTrackingInfo>
<DocTitle>
  ⋮
</DocTitle>
<Authors>
  ⋮
</Authors>
  ⋮
</DocTrackingInfo>
<PropertyDefinitions>
<PropertyDef ...>
  ⋮
</PropertyDefinitions>
<LinkDescriptions>
<LinkDesc id=lnk0123 ...>
  ⋮
</LinkDescriptions>
  ⋮
</prolog>
<FrontMatter>
  ⋮
<body>
  ⋮
</IBMIDDOC>
```

SGML Definition

```
<!ELEMENT  Prolog           - O (%Prolog;) >
<!ATTLIST  Prolog
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Metadata"
>
```

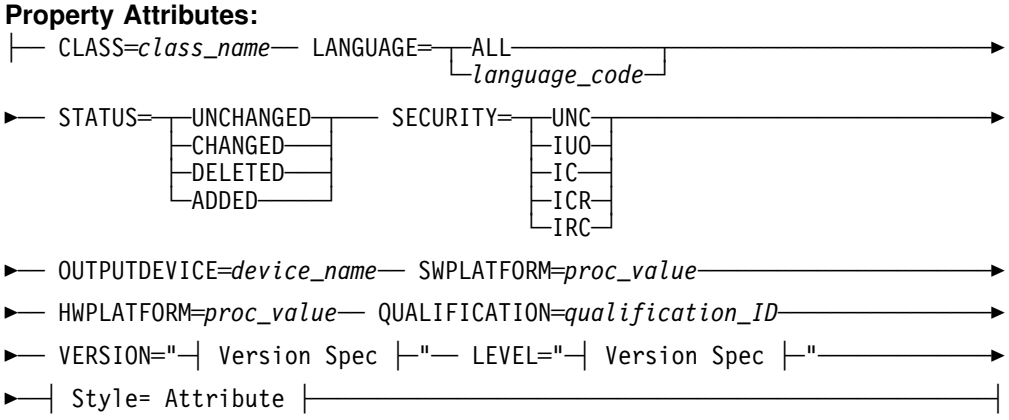
Property Attributes (element properties)

Purpose

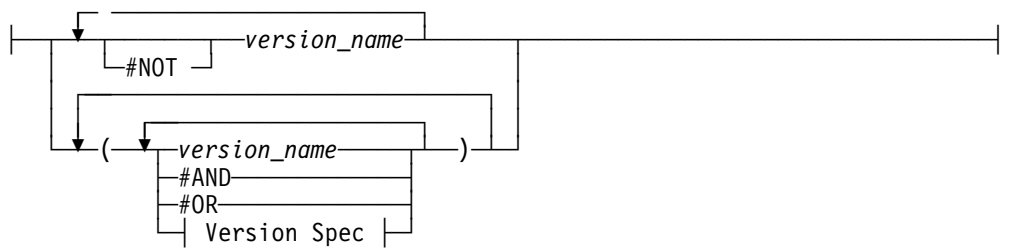
These attributes define IBM- and IBM ID Doc-specific property attributes usable on any element. All property values are inherited by child elements from their parents. In addition, properties can be inherited by direct reference from any element to a PropertyDef element. Multiple inheritance is supported for all properties.

Syntax

Property Attributes:



Version Spec:



Attributes and Contained Elements

LANGUAGE=*language_code*

Defines the language that a given element is written in. The keyword **#ALL** indicates that the element applies to all languages.

Note that the element content must still be coded in the document character set regardless of the language value. In other words, unlike BookMaster, the char-

acter encoding of the data is independent of the human language of the text itself. If double-byte characters and special characters are not included in the document character set, they will be interpreted according to the document character set description even when the language typically uses double-byte encoding. SGML does not allow dynamic changing of character sets within a document.

STATUS=*status value*
Indicates the change status of the element, one of CHANGED, NEW, DELETED, or UNCHANGED. STATUS= is required when a RevisionSpecification hyperlink is defined for the element.

QUALIFICATION=*qualification_id*
Refers to a Qualification element in the document or division prolog that defines the revision that applies to the element.

SECURITY=*security_code*
Defines the IBM-specific security classification of the element. Any element's security classification is the highest of its classification and the classification of any elements it contains. Retrieval alternatives and elements accessed via ContentRef= are considered to be contained, regardless of where the element source is specified.

OUTPUTDEVICE=*device_name*
Indicates the output devices or processes to which the element applies. OUTPUTDEVICE= can be used to indicate that a given element should only be used by a specific output processor, for example, only for print or only for online.

SWPLATFORM=*platform_name*
Indicates the software platforms (operating system or operating environment) to which the element applies. Typical values would be `'OS/2'`, `'VM'`, and `'Windows'`.

The SWPlatform= value can be used to keep information for different specific software platforms in the same document.

HWPLATFORM=*platform_name*
Indicates the hardware platforms (processor architectures or machine names) to which the element applies. Typical values would be `'RISC/6000'`, `'370'`, and `'80x86'`.

The HWPlatform= value can be used to keep information for different specific hardware platforms in the same document.

VERSION=*version_spec*
LEVEL=*version_spec*
Version= defines what author-defined versions of the document the element applies to. Level= defines what levels of a document an element applies to. In normal usage, versions represent different versions of a document at a given release level, for example, a full reference and a reference summary, or different operating system versions of a user's guide. Levels normally represent different release levels of a single version.

The Version= and Level= attributes let you specify conditional selection of arbitrary complexity by your choice of version names and the use of logical operators to create logical expressions. Active versions are defined to the IBM ID Doc processor at the time the document is processed.

You can use the logical operator keywords **#AND**, **#OR**, and **#NOT** together with version and level names to create Boolean expressions as complex as you need them to be. You can use parenthesis to group expressions. Expressions are evaluated from left to right.

Usage

Each of these properties defines some intrinsic aspect of the element, such as its security classification,

Examples

Simple example showing language and security specification:

```
<IBMIDDOC language=English security="IC">
```

Example showing the use of VERSION= and XVERSION=:

```
<div version="(v1r1 #OR v1r2)">Use this division when
either V1R1 or V1R2 is the version.
<p version="#NOT v1r2">This paragraph should not be used
when V1R2 is the version.
```

SGML Definition

```
<!ENTITY % IBMProperty.Attributes
   "IBMSecurity      (UNC | IUO | IC | ICR | IRC)
                                  #IMPLIED"
>

<!ENTITY % Property.Attributes
   "Language         NAME         #IMPLIED
    Security         NAME         #IMPLIED
    Status           (New | Changed | Deleted | Unchanged)
                                  Unchanged
    Style            CDATA        #IMPLIED
    Version          CDATA        #IMPLIED
    Level            CDATA        #IMPLIED
    HWPlatform       CDATA        #IMPLIED
    SWPlatform       CDATA        #IMPLIED
    OutputDevice     CDATA        #IMPLIED
    %IBMProperty.Attributes;"
>

<!-- Definition of VersionSpec model

<HyLex ltn=VersionSpec>
     S,
     (("#NOT", S)?, name, S) |
     (VersionSpec, ("#AND" | "#OR"), VersionSpec) |
     ("(", VersionSpec, ")", S)
</HyLex>

-->
```

PropertyDef (property set definition)

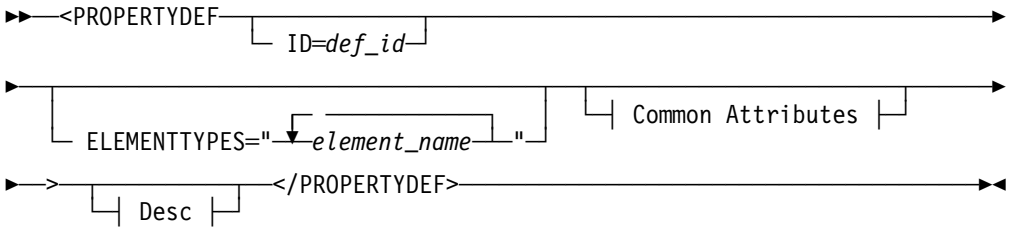
Purpose

Defines property values for properties common to all elements. Any element can refer to a PropertyDef element using the common PropertyRef= attribute. When a set of properties is meaningful only for a certain set of element types, the ElementTypes= attribute can be used to indicate which element types can refer to a given PropertyDef element.

Valid Contexts

PropertyDefinitions, PropertyGroup.

Syntax



Attributes and Contained Elements

ID=*def_id*

Property definition elements without an ID= define the default properties for the division or document within which they occur. Property definition elements that do not include an ElementTypes= attribute and that do not have an ID= value will apply to all elements. When an ElementTypes= value is specified, the PropertyDef values will apply only to elements of the specified type.

When there are multiple PropertyDef elements without ID= values, the last property specified will be the one used, with the PropertyDef elements processed in the order they occur in the document source. You can use PropertyGroup elements to organize PropertyDef elements in order to provide different sets of property values for different conditions.

ELEMENTTYPES=*element names*

Defines those element types (generic identifiers) to which this PropertyDef applies. Use ElementTypes= when a PropertyDef is only meaningful for a specific set of element types, such as when the Style= value is element specific.

Desc

An optional text description of the property definition. This is intended to provide documentation about why a certain set of properties have been defined.

Usage

Use PropertyDef to define values for property attributes that are common to all elements, such Language=, Proc=, and Style=. For example, you can define a complex set of properties for doing conditional processing using PropertyDef and then use those values from other elements by refering to the PropertyDef element, rather than having to explicitly specify those attributes on each element.

Examples

```
<prolog>
  ⋮
<PropertyDefinitions>
 <PropertyDef version='v2r3'>
 <ClassDef ElementType=figure classname=codesamples>
  <desc>Figures that contain code samples
 </ClassDef>
</PropertyDefinitions>
  ⋮
</prolog>
```

SGML Definition

```
<!ELEMENT  PropertyDef      - O (desc?)         -- Property Defintion -->
<!ATTLIST  PropertyDef
    ElementTypes     NAMES        #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "PropertyDefinition"
>
```

PropertyDefinitions

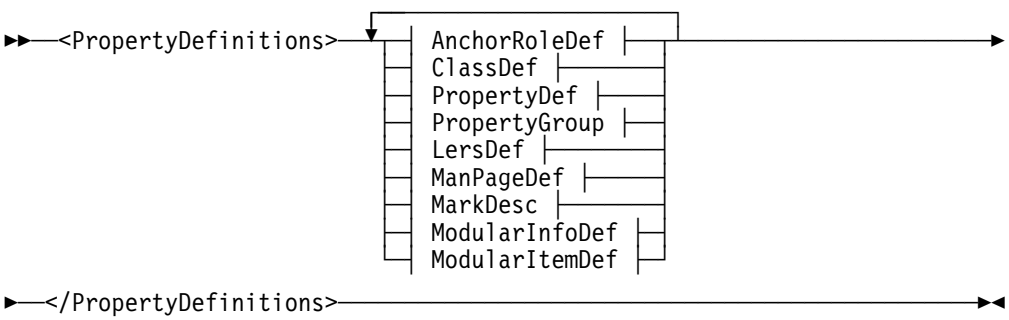
Purpose

Contains property definition elements. Properties apply to all elements contained within the division with which the property definitions are associated.

Valid Contexts

Prolog, DivProlog.

Syntax



Attributes and Contained Elements

Definition Elements

All of the property definition elements are valid within PropertyDefinitions.

Usage

Use PropertyDefinitions to contain property and class definitions for elements within a document or division. Property definitions that do not have an ID= attribute specified define the default properties for the element type they are defined for.

The property values specified on property definitions that do have an ID= attribute specified only take effect when those property definition elements are explicitly refered to from another element.

The PropertyDef element defines values for properties that are common to all elements.

Examples

```
<prolog>
  ⋮
<PropertyDefinitions>
 <PropertyDef version='v2r3'>
 <ClassDef classname=codesamples ElementType=figure>
  <semantic>Figures that contain code samples.
 </ClassDef>
</PropertyDefinitions>
  ⋮
</prolog>
```

SGML Definition

```
<!ELEMENT  PropertyDefinitions
                               - - (%PropertyDefinitions;)*
>
<!ATTLIST  PropertyDefinitions
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

PropertyGroup

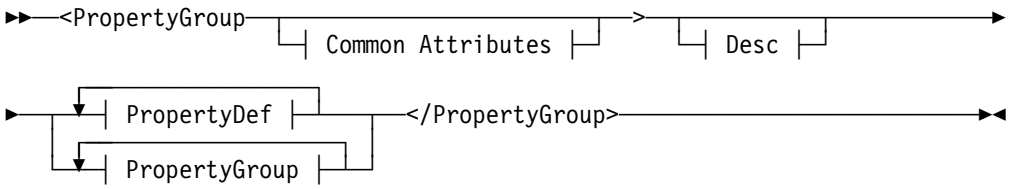
Purpose

The PropertyGroup element lets you organize PropertyDef elements within a PropertyDefinitions section and use properties to control the use of different sets of PropertyDefs.

Valid Contexts

PropertyDefinitions, PropertyGroup.

Syntax



Attributes and Contained Elements

Common Attributes
Property attributes specified on PropertyGroup will be used by the processing system to determine whether or not a given PropertyGroup will be processed.

Desc
An optional Desc element, intended to contain an explanation of the purpose of a given PropertyGroup element.

PropertyDef
One or more PropertyDef elements.

PropertyGroup
PropertyGroup elements can be nested within other PropertyGroups in order to use inheritance of properties.

Usage

Because PropertyDef elements are used to define property values, the processing system cannot then use those same properties to determine whether or not a given PropertyDef element should be processed. The PropertyGroup provides a way to use element properties to control processing of PropertyDef elements.

The typical case is one where you want one set of properties for one output and a different set for another. You can use PropertyGroup elements with the Proc= attribute to contain the different PropertyDef elements.

You can nest PropertyGroups in order to take advantage of property inheritance.

Examples

```
<prolog>
  ⋮
<PropertyDefinitions
<propertydef security="IUO" version="expert">
 <desc>This property def applies to all elements in the document
</propertydef>
<PropertyGroup proc=display>
<desc>The following property definition only applies
when processing for online display.
</desc>
<propertydef ...>
</PropertyGroup>
<PropertyGroup proc=print>
```

```
<desc>The following property definition only applies
when processing for print.
</desc>
<propertydef ...>
</PropertyGroup>
</PropertyDefinitions
  ⋮
</prolog>
```

SGML Definition

```
<!ELEMENT  PropertyGroup    - - (Desc?, (PropertyDef* | PropertyGroup*))>
<!ATTLIST  PropertyGroup
    %Common.Atts;
>
```

PropertySource Anchor Role (property reference link)

Purpose

When specified as a value of the AnchorRoles= attribute, indicates that the cooresponding Linkends= ID value refers to a PropertyDef element from which to inherit properties.

Usage

Use the PropertySource role in conjunction with Linkends= to refer to a PropertyDef element in order to use its property values. Property values obtained by property reference override any properties inherited from an element's parent. Property-reference-defined properties can themselves be overridden by specifying them directly on the element.

Migration Note

Property reference links replace the DCF BookMaster REFID= attribute when it refered to a DEF element. In addition, now any element can use PROPERTYREF, not just specific elements.

Examples

Simple example showing reference from a figure to a property definition:

```
<PropertyDef id=figdef1 style='inline'>
⋮
<figure linkends   ="figdef1" HyTime=ilink
        anchorroles="linksource propertsource">
```

Note that the first role name, linksource, refers to the Figure element itself, indicating that it is the source of the link.

PublicIdentifier

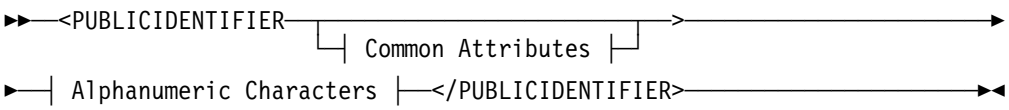
Purpose

Contains the SGML public identifier for a document.

Valid Contexts

BibRecord, IBMBibRecord, LibRecord, IBMLibrecord.

Syntax



Attributes and Contained Elements

Alphanumeric Characters
The text of the public identifier itself. For IBMIDDoc documents, this must be a formal public identifier.

Usage

It is intended that the public identifier of the document entity be used by presentation systems to locate the actual document, but specific presentation systems may define application-specific data to be specified as the system identifier of the document entity if they do not support the use of public identifiers. The public identifier can be included in the BibRecord itself as a way of keeping a document's formal public identifier definition with the rest of its bibliographic information. This could allow, for example, the automatic generation of entity declarations for documents described by BibRecord elements.

See Appendix A, “Proposed IBM Standard for Formal Public Identifiers” on page 332 for a description of the formal public identifier standard defined by the IBM Corporation for its internal and external use.

Examples

```
<bibrecord id=iddocref>
  ⋮
 <publicidentifier>+//ISBN 0-933186::IBM//DOCUMENT IBMIDDoc Reference//EN
  ⋮
</bibrecord>
```

SGML Definition

Quote (quotation phrase)

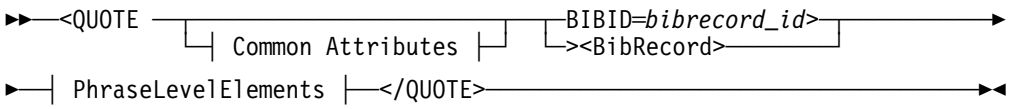
Purpose

Quote contains a quoted phrase within the context of a paragraph or similar element. Quote associates a quotation with the bibliographic information about its source.

Valid Contexts

PhraseLevelElements.

Syntax



Attributes and Contained Elements

BIBID=*bibrecord_id*

The ID of the BibRecord element that defines the source of the quotation. You must either specify BibID= or include a BibRecord element within the Quote element.

BibRecord

Contains the bibliographic information for the source of the quotation.

Usage

Use the Quote element to contain material excerpted from another source and that is used within the context of a paragraph or similar element. Use LongQuote for quotations are themselves more than one pargraph. In the default presentation style, the quoted material is presented inline with the material around it. It is usually surrounded by typographical quotes as well.

Because the purpose of Quote is to identify a quotation, it must either refer to a BibRecord element using the BibID= attribute or directly contain a BibRecord element.

Do not use Quote to simply surround a phrase with typographic quotation marks. Instead use the &otq; (open typographic quote) and &ctq; (close typographic quote) entities. The IBM ID Doc processor will produce the proper typographic quote characters, including accounting for nested quotation marks.

Examples

```
⋮
<p>New presidents often try to inspire the country to face new challenges
with words like:
<quote bibid=jfk0161>Ask not what your country can do for you,
ask what you can do for your country</quote>
⋮
```

SGML Definition

```
<!ELEMENT  Quote            - O (%DataPool;)* >
<!ATTLIST  Quote
    BibID           NAME         #IMPLIED
    %Common.Atts.NoRefType;

    RefType         CDATA        #FIXED
                         "BibID (&BibRecord;)
                          &Common.RefType;"
    InfoMaster      NAME         #FIXED    "Excerpt"
>
```

Range (syntactic range)

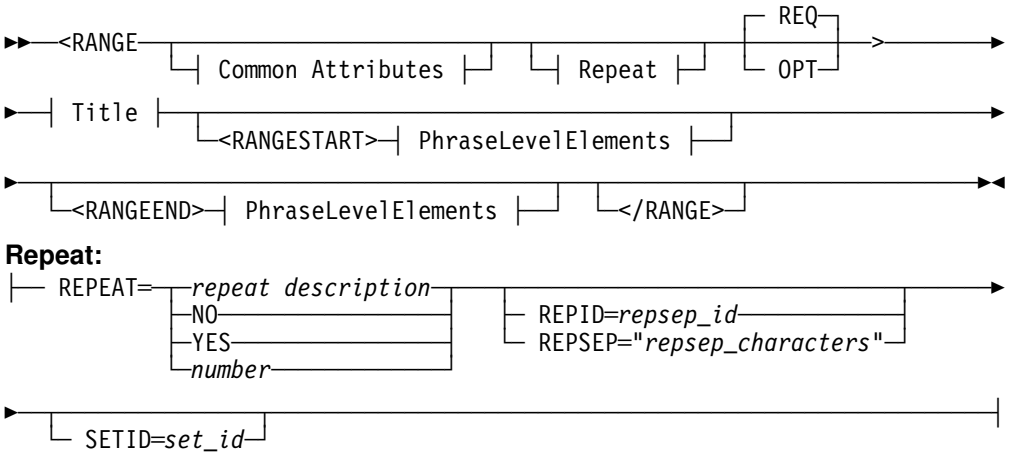
Purpose

Within a syntax definition, defines a choice as a range of values, either defined directly or by reference to a previously-defined set. Range is a specialized form of Group.

Valid Contexts

Fragment, Group, RepSep, SyntaxBlock, SyntaxPhrase, Syntax.

Syntax



Attributes and Contained Elements

REPEAT=*repeat*

Specifies whether or not the choice represented by the range is repeatable and if so, how many times. The specification can be:

YES

Indicates that the group is repeatable with no limit.

NO

Indicates that the group is not repeatable (the default).

integer
Indicates the number of times the group can be repeated.

repeat description
A prose specification of the stop rule, for example, "choose one or more items. Do not repeat any items.".

REPID=*repsep_id*
The ID of a REPSEP element to be used as the repeat separator. The repeat separator can also be specified with the REPSEP= attribute.

REPSEP=*separator_value*
Specifies a simple repeat separator. This value is treated as if it were specified as a SEP element. REPSEP= and REPID= are mutually exclusive.

REQ | **OPT**
Indicates whether or not the range is optional. REQ (required) is the default.

SETID=*set_id*
The ID of a set that defines the domain of the range. When no RangeStart or RangeEnd elements are specified, the range is the entire set, otherwise the range is a subset of the range as defined by the RangeStart and RangeEnd elements.

Title
The title of the range.

RangeStart
RangeEnd
Define the start and end of a range. When the range refers to a set, RangeStart and RangeEnd define the subset of the referenced set to be used for the range. If the range does not refer to a set, you must specify values for RangeStart and RangeEnd.

When the range refers to a set, IBM ID Doc does not check to ensure that the values for RangeStart and RangeEnd are members of the set, but a system that uses the markup for purposes other than presentation may perform that check.

Usage

Use Range where you would otherwise use a choice group to indicate a choice among a large set of items or where the set from which the choices can be made is defined by description rather than defined explicitly. For example, if the choices were the months of the year, you could either create a choice group with each month specified as a keyword, or you could use a range which refers to a set that defines the months of the year. With Range, the presentation system can present the range in a variety of ways, for example, showing the first and last elements of the range or showing the whole range, if appropriate, where as with a choice group, the entire group must be shown.

Another use for Range is to indicate that the choice is from a large or even infinite range, for example, the set of whole numbers.

When you specify a range, you can either refer to a set or simply specify the start and end of the range. If you do not refer to a set, then you must specify both the start and end of the range. If you do refer to a set, then you can specify the start

or end of the range (or both). For example, you could indicate the range of the months of the year by simply specifying the start and end of the range, like so:

```
<range>Months of the Year
  <rangestart>January
  <rangeend>December
</range>
```

You can refer to a set like so:

```
<range setid=NonZeroIntegers>Non-zero integers
</range>
```

Given the set 'NonZeroIntegers', you could define the range of numbers from 1 to 100 like so:

```
<range setid=NonZeroIntegers>
 <rangestart>1
 <rangeend>100
</range>
```

Examples

```
<syntax>Non-Zero Integers
 <set id=NonZeroInts>non-zero integers
 </set>
 <range setid=NonZeroInts>
  <rangestart>1
  <rangeend>100
 </range>
</syntax>
```

SGML Definition

```
<!ELEMENT  Range            - O (Title, RangeStart?, RangeEnd?) >
<!ATTLIST  Range
    Repeat           CDATA        #IMPLIED
    RepID            NAME         #IMPLIED -- Must be REPSEP ID --
    SetID            NAME         #IMPLIED -- Must be SET ID --
    RepSep           CDATA        #IMPLIED -- Repeat separator chars --
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                          "REPEAT ('YES' | 'NO' | number | rcdata)
                           &Common.LexModel;"
    RefType          CDATA        #FIXED
                          "REPID REPSEP SETID SET &Common.RefType;"
>

<!ELEMENT  (RangeStart | RangeEnd)
                             - O (%DataPool;)* >
<!ATTLIST  (RangeStart | RangeEnd)
    %Common.Atts;
>
```

RefKey (reference key hyperlink)

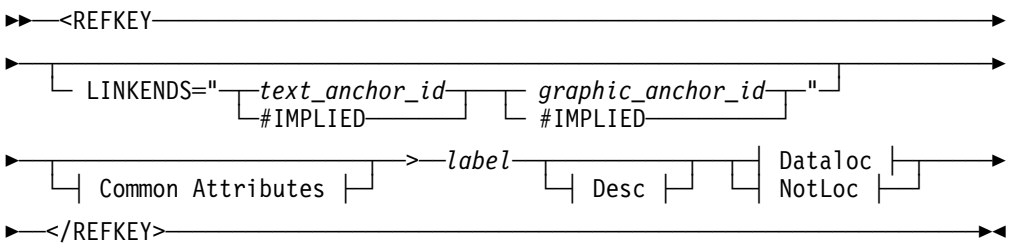
Purpose

Creates a labeled link between text elements and non-text elements. When used with explicitly-specified Linkends= values, RefKey conforms to the HyTime ilink element form.

Valid Contexts

PhraseLevelElements, ProcStep.

Syntax



Attributes and Contained Elements

LINKENDS=*text_anchor_id graphic_anchor_id*

Defines the text and graphic that the reference key connects together. When the RefKey occurs within the context of the text element to be linked, the keyword **#IMPLIED** can be specified for the text anchor ID. Likewise, when the RefKey occurs within the context of the graphic element to be linked, the keyword **#IMPLIED** can be specified for the graphic anchor ID.

label

The text to be used as the reference key label. This label will be used in the text and possibly in the graphic to visually indicate the association between the text and the graphic. Online presentation systems may choose to represent reference key links in other ways as well, for example, as special icons or boxed selection areas in graphics.

Desc

An optional description of the reference key. Normally intended to provide information about why a given reference key link was defined.

NotLoc | **Dataloc**

The Notation-specific or data location element that defines the location of the graphic link. Notloc and Dataloc elements can also be contained by LinkDesc elements.

Usage

Reference keys are used to connect text and graphics (or other non-text objects, such as online tutorials or active programs) using hyperlinks. However, reference keys can also be used without specifying any link anchors if all you need is a visual reference to a reference key.

Reference key links have two ends, or *anchors*. The first anchor is always within a text element. The second anchor is always within a graphic or other non-text object. Anchors are refered to directly by their ID values, or indirectly via a LinkDesc element. You could also use a LinkDesc to define multiple text anchors by using the text anchor to refer to a LinkDesc that then refers to multiple text elements. Finally, reference keys can be linked to from a text or graphic element. For example, a procedure step may refer directly to a reference key element rather than requiring that the reference key be specified within the step. This is useful when the same point within a graphic is refered to many times.

Reference keys that are to be re-used in this way or that define links to graphics that may change in the future can be collected in an object library and refered to from anywhere within the document.

Note: Reference Key is a specialized instance of the HyTime ilink architectural form.

Examples

Simple example with both link anchors explicitly specified:

```
<refkey linkends="paragraph1 graphregionA">A</refkey>
```

Reference key within text context:

```
<p>Refer to
<refkey linkends="#implied graphregionA">A</refkey> in
<xref refid=myfigure>.
```

In this example, the text anchor is defined implicitly by the RefKey element's placement in the source.

The next example shows reference keys within an object library to be used by reference from elsewhere. Note that because they will be refered to from within the text, the text anchor of each reference key is implied. Within each RefKey is a NotLoc element that defines the region within the graphic that the RefKey element is linked to. The LOCSRC= attribute on NotLoc indicates the object that the NotLoc element applies to, which in IBM ID Doc is always a data entity (as opposed to another element elsewhere in the document). Because the RefKey element directly contains the NotLoc element, the graphic anchor is also defined implicitly. Because both the text and graphic anchors are implied, there is no need to specify the LinkEnds= attribute in this example.

```
<objectlibrary id=mygraphicrefkeys>
<desc>Contains reference keys that link to regions within graphic entity
MyGraphic.
Each graphic anchor is defined by the NotLoc element contained within
each RefKey.
The anchors of the RefKeys are thus both implied, the text
anchor being implied by the position of the RefKey in the
source, and the graphic anchor defined by the NotLoc, which
is implicitly linked to the RefKey because the RefKey element
contains it.
defining the graphic area being linked to:
</desc>
<!-- Definition of reference key A -->
<refkey id=refkeyA>A
<desc>Note the position of the mouse pointer in this window.</desc>
<notloc locsrc=mygraphic>ulcorner=(100,100) lrcorner=(200,200)</notloc>
</refkey>
<!-- Definition of reference key B -->
<refkey id=refkeyB>B
<desc>Note the folder icon for the active object</desc>
<notloc locsrc=mygraphic>
ulcorner=(400,400) lrcorner=(500,500)</notloc>
</refkey>
</objectlibrary>
```

NotLoc elements can also be used within LinkDesc elements, so you could use a LinkDesc element to contain the NotLoc and then refer to the LinkDesc as the graphic anchor. You might want to do this if you will be using the graphic link for purposes other than RefKey links. For example, the same point might also be a hot spot within the graphic. The following example is the same as the above, except that the NotLoc elements are now within LinkDescs rather than within the RefKey elements themselves:

```
<linkdesc id=graphicanchorA>
 <desc>Defines a link anchor within a bitmapped graphic.
 <notloc locsrc=mygraphic>ulcorner=(100,100) lrcorner=(200,200)</notloc>
</linkdesc>
  ⋮
<refkey id=refkeyA linkends="#IMPLIED graphicanchorA>A
<desc>Note the position of the mouse pointer in this window.</desc>
</refkey>
```

SGML Definition

```
<!ELEMENT  RefKey           - - (RefKeyBody, desc?, NotLoc?)
                                                  -- Reference Key --
>
<!ATTLIST  RefKey
--  LinkEnds         NAMES        #IMPLIED  part of common attributes --
    %Common.Atts.NoLexMdl.NoHyTime;

    anchrole         CDATA        #FIXED    "textanchor graphicanchor"
    LexModel         CDATA        #FIXED
                          "LinkEnds ( (idref | '#IMPLIED'),
                                      (idref | '#IMPLIED'))
                           &Common.LexModel;"
    HyTime           NAME         #FIXED    "ilink"
```

```
                                                -- HyTime independent link form --
>

<!ELEMENT  RefKeyBody       O O (%DataPool;)*   -- Reference Key Body -->
<!ATTLIST  RefKeyBody
    %Common.Atts;
>
```

RepSep (repeat separator)

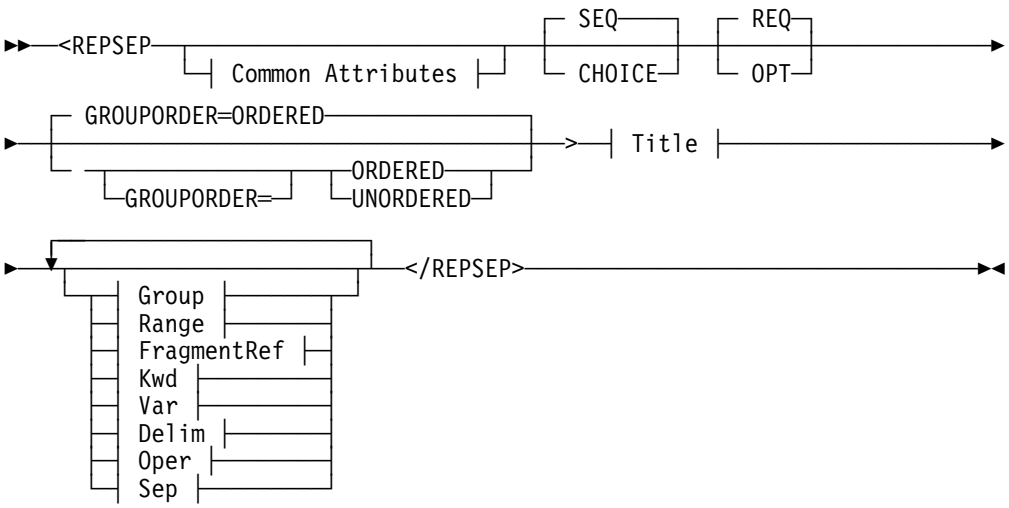
Purpose

Defines a repeat separator within a syntax definition. The repeat separator can be of arbitrary complexity. RepSep is a special case of Group.

Valid Contexts

Fragment, Syntax, SyntaxBlock, SyntaxPhrase.

Syntax



Attributes and Contained Elements

SEQ | CHOICE
Indicates whether or not the repeat separator is a set of choices. Sequential (SEQ) is the default.

OPT | REQ
Indicates whether or not the Any group that is not optional is by definition required.

GROUPORDER=ORDERED | UNORDERED
Indicates whether or not the order of elements within the repeat separator is important. The default is **ORDERED**.

Title
The title of the repeat separator. The repeat separator title is used as the fragment name should the repeat separator need to be turned into a fragment in the event the repeat separator is too wide to fit in the presentation space.

Group
Repeat separators can themselves contain groups. The nesting depth is limited only by the SGML TAGLVL value as defined in the SGML declaration.

Range
Defines a choice among a range of values as defined in a set.

FragmentRef
A reference to a syntax fragment.

Kwd
Contains a literal keyword.

Var
Contains a syntactic variable.

Delim
Contains a syntactic delimiter. Delimiters are normally containing in that they have a start and an end, as opposed to separators, which serve to separate subelements within a statement. Typical delimiters are quotation marks and parentheses.

Sep
Contains a syntactic separator. Separators separate one subelement of a statement from another. Typical separators are significant blanks and commas.

Oper
Contains a syntactic operator. Operators normally connect two elements together and specify an operation performed on one or both elements. Typical operators are the equals sign (=) and the plus sign (+).

Usage

Use RepSep to define repeat separators that are more than just a single keyword. Repeat separators must be defined at the beginning of the syntax definition.

Examples

```
<repsep>Group Element
  <delim start>&stago;
  <kwd>GROUP
  <delim start>&tagc;
    <fragmentref repeat="yes">Group Elements
  <delim start>&etago;
  <kwd>GROUP
  <delim start>&tagc;
</repsep>
```

SGML Definition

```
<!ELEMENT  RepSep           - - (Title, (%GroupElem;)*)
                               -- Separator for repeated syntax tokens --
>
<!ATTLIST  RepSep
    OptReq           (Opt | Req| Def)  Req
    CHOICESEQ        (Seq | Choice)
                                  Seq
    GROUPORDER       (ordered | unordered)
                                  Ordered
    %Common.Atts;
>
```

Alternate Anchor Role (retrieval alternative link)

Purpose

When specified on the common AnchorRoles= attribute, indicates that the corresponding Linkends= ID value represents alternative versions of the element making the reference.

Usage

Use the alternative anchor role to refer to alternate versions of an element that have different property values. For example, you can link together different language versions of the same element. Only those versions that match the current language setting will be used (or “retrieved”). Alternative versions of an element can be specified together or object libraries can be used to isolate alternative versions.

When specifying retrieval alternatives, you can use the RetrieveAs= attribute to indicate whether or not the element making the reference should be considered as the default version or as just another alternative. Retrieve as default is the default value for RetrieveAs=. When default is specified, the element making the reference will be used only if none of the alternatives are used, and will not be used if any of the alternatives are used. When RetrieveAs= is set to **alternate**, the element making the reference will be used if it matches the current property state, and may be used in addition to any other alternates that also match the current property state.

Set RetrievaAs to **alternate** when you need to enable the retrieval of multiple alternate versions as once, such as in a side-by-side presentation of different language versions in a document. When **alternate** is specified, you should ensure that the various property values will ensure that at least one of the alternates will be retrieved, unless it is acceptible to have the element completely suppressed under some conditions.

Examples

```
<p id=para001 language=english
   linkends="lang-alts-para001"
   anchorrole="linksource alternate"
   retrieveas="alternate">
<nameloc id=lang-alts-para001>
 <nmlist>french-para-001 german-para-001 japanese-para-001</nmlist>
</nameloc>
This paragraph has three other translations available for it.  They are
each in a different object library, one for French, one for
German, and one for Japanese.
```

Revision

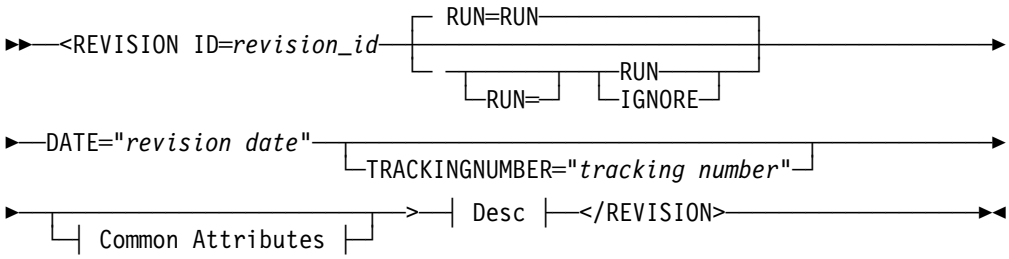
Purpose

Revision defines a single revision for a document or division. Elements can use the common Revision= attribute to refer to a Revision element in order to indicate that they have been revised and what revision the change applies to.

Valid Contexts

RevisionHistory.

Syntax



Attributes and Contained Elements

ID=*revision_ID*

The ID of this revision. The ID= attribute is required. The ID= is refered to with the common Revision= attribute from any element.

RUN
IGNORE

Indicates whether the revision is an active revision. Revisions that are active (RUN) will be indicated by whatever mechanism is defined in the presentation style, typically by placing a vertical bar or other character in the margin.

Revisions for which IGNORE is specified are ignored and knowledge of that revision is preserved in any way when the document is processed.

DATE=*revision date*

Indicates the date of the revision. This date can be a date range or an indication of the time frame to which the revision applies, such as “first draft.”

TRACKINGNUMBER=*tracking number*
Associates the revision with some revision tracking mechanism such as a change request or problem number.

Desc
A description of the revision.

Usage

Use Revision to define a single revision to a document or division. You can define as many different revisions as you need to. Revisions are refered to from any element using the common Revision anchor role.

Use the Desc element within Revision to describe the purpose of the revision. Because Desc can contain any phrase- or paragraph-level element, you can use elements to structure the description information. For example, you might use a standard definition list to organize information needed by your local processes in tracking revisions.

Examples

```
<Prolog>
  ⋮
<RevisionHistory>
<revision id=draft1comments date='12/21/92'>
<desc>Changes to the first draft as a result of the
first draft review.
</RevisionHistory>
  ⋮
</Prolog>
```

SGML Definition

```
<!ELEMENT  Revision         - O (Desc, Author, Date)
                                          -- Revision definition --
>
<!ATTLIST  Revision
    ID               NAME         #REQUIRED
    TrackingCode     CDATA        #IMPLIED  -- Revision tracking code --
    Reason           CDATA        #IMPLIED
                    -- Reason for revision: editorial, technical, etc. --
    %Common.Atts.NoID;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

RevisionHistory (revision tracking information)

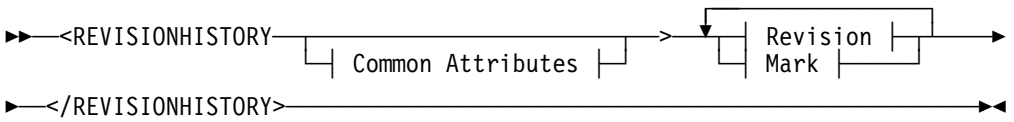
Purpose

RevisionHistory contains Revision and Mark elements within a Prolog or DivProlog and defines the revision history of the document or division.

Valid Contexts

DocTrackingInfo, TrackingInfo.

Syntax



Attributes and Contained Elements

Revision
One or more Revision elements.

Mark
One or more Mark elements.

Usage

Use RevisionHistory to contain the Revision elements that define the revisions applicable to a given document or division.

Examples

```
<Prolog>
  ⋮
<RevisionHistory>
<revision id=draft1 run date='12/21/92'>
Changes for draft one
</revision>
</RevisionHistory>
```

SGML Definition

```
<!ELEMENT  RevisionHistory  - - (Revision | Mark)+ >
<!ATTLIST  RevisionHistory
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

RevisionSpecification Anchor Role (revision specification link)

Purpose

When specified as a value of the AnchorRoles= attribute, indicates that the cooresponding Linkends= ID value refers to a Revision element.

Usage

Use the RevisionSpecification role in conjunction with Linkends= to refer to one or more Revision elements in order to indicate the revisions that apply to this element.

When the RevisionSpecification role is used, you must also specify the Status= property attribute to define the revision status of the element.

Examples

```
<p linkends   ="para-1-revs" HyTime=ilink
   anchorroles="linksource RevisionSpecification" status=changed>
<nameloc id=para-1-revs>
 <nmlist>revision-11-93 revision-12-93</nmlist>
</nameloc>
This paragraph has been
revised.
```

Note that the first role name, linksource, refers to the paragraph element itself, indicating that it is the source of the link.

Safety (safety notices)

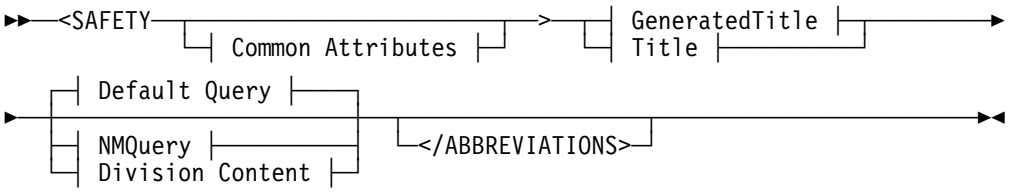
Purpose

Safety is intended to contain any saftey-related notices in the information.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the Safety division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

NMQuery
Default Query
The content of the Safety division can be generated by a query. If no division content is specified for the Safety element, then the default query will be used. Alternatively, you can specify an explicit query to generate the content.

The suggested default query is to locate all safety-related elements in the document.

Division Content
Safety has the same content rules as Div elements.

Usage

Use Safety to contain or refer to any safety-related information such as cautions, warnings, FCC notices, and the like. If your system supports it, you may be able to use queries to automatically generate the presentation of safety notices used elsewhere in the information. Conversely, you can collect safety information within Safety and use it by reference in other locations as needed.

Examples

```
<frontmatter>
<safety><generatedtitle>
<caution>Use of the following commands may lead to data loss:
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  Safety           - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool;)*)
                                )
>
<!ATTLIST  Safety
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>
```

Screen (computer screen or panel)

Purpose

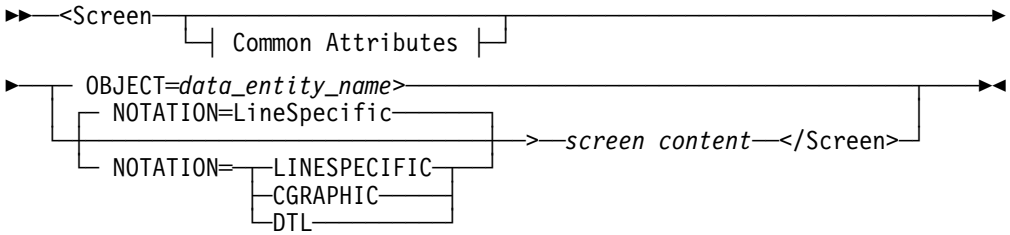
Contains or refers to a representation of a computer screen or user interface panel (window).

Screen (computer screen or panel)

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

OBJECT=*data_entity_name*

Refers to an SGML data entity that contains the screen representation. The data entity can be in any supported notation and may in fact be a reference to a live version of the panel if that is supported by your online presentation system.

When Object= is specified, it is an error to specify any screen data or the Screen end tag.

NOTATION=*notation name*

For inline screen data, defines the notation of the contained data. Supported notations are:

LINESPECIFIC

Significant record ends are preserved in the output.

CGRAPHIC

The data defines a box and line graphic using characters.

DTL

ISPF dialog tag language panel definition. When coded inline, the ISPF panel must be contained within a CDATA marked section. The better method is to define a CDATA entity for the DTL panel and refer to the entity with the Object= attribute. The panel will be processed by a separate DTL processor for the purpose of creating a representation of the panel for the presentation device.

screen content

The inline screen data. The screen data can consist of any phrase-level elements. You can also use MultiMediaObject or ObjectContainer elements to refer to external data entities that contain the screen data (such as screen capture graphics, for example). Dialog Tag Language markup, or any other SGML or SGML-like markup, must be contained within a CDATA or RCDATA marked section.

Usage

Use Screen to contain representations of online panels, windows, or other user interface components. The default print presentation style is to enclose the screen representation within a box with rounded corners, suggesting a computer display screen.

Examples

```
<figure id=screen1><caption>Sample Dialog Tag Language Screen
<Screen notation=dtl>
<![ CDATA [
<panel id=panel1>
  ⋮
</panel>
]]>
</screen>
</figure>
```

SGML Definition

```
<!-- DTL notation data is referenced using the Object attribute -->
<!ENTITY % Screen.Notations "LineSpecific | CGraphic" >

<!ELEMENT  Screen           - - (%DataPool;)* >
<!ATTLIST  Screen
    Object           ENTITY       #CONREF
    Notation         (%Screen.Notations;)
                                  LineSpecific
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    ApplicationClass NAME         #FIXED    "Screen"
>
```

Set (syntactic set)

Purpose

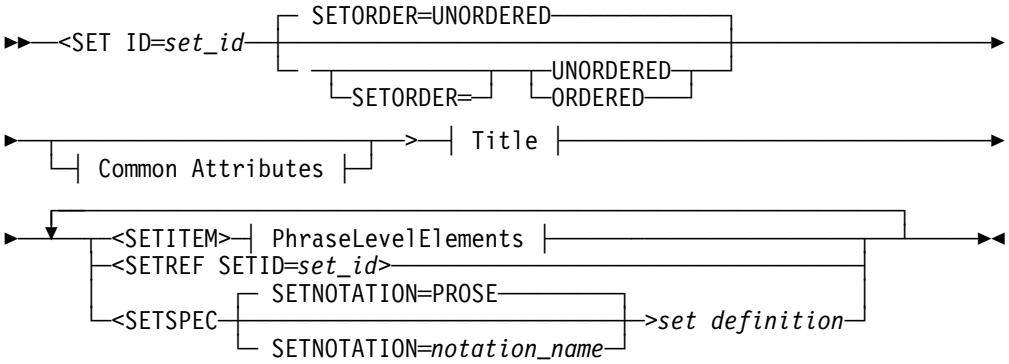
Defines a set used either to further define a variable or used as the base for a range specification.

Valid Contexts

Syntax, SyntaxBlock.

Syntax

Set (syntactic set)



Attributes and Contained Elements

ID=*set_id*
The ID for this set, used to refer to the set from syntactic elements.

SETORDER=ORDERED | **UNORDERED**
Defines whether or not the set is ordered. If ORDERED is specified, the order of the SetItem elements determines the order of the elements within the set. The default is **UNORDERED**.

Only ordered sets can be used to define ranges.

Title
The descriptive title of the set. This is the text that will be used in references to the set or when the set definition is presented.

SETITEM
Contains the value of a single set item. Set items are always literal strings.

SETREF SETID=*set_id*
A reference to another set. SETREF allows you to build a set from other sets.

SETSPEC
set definition
Defines an algorithmic specification of the set. This can be in any notation desired, including simply a prose description of the set, for example:

```
<setspec setnotation=prose>All prime numbers except 1</setspec>
```

The SETNOTATION= attribute indicates the notation used to describe the set. This is not interpreted as an SGML notation reference, although a processing application may use the SETNOTATION= value to do special processing on the SETSPEC content. It is not necessary to declare SETNOTATION values as SGML notations, although you can. IBM ID Doc supports the use of HyLex notation for defining sets. The default notation is **PROSE**.

Usage

Use SET definitions to further refine the definition of variables or to provide a base for defining ranges. For example, you could use sets to define the set of characters valid in filenames or use a SET definition and a RANGE specification to define a choice among a large set of parameters. Sets used to define ranges must be ordered sets.

Examples

```
<set id=nzintegers ordered>Non-Zero Integers
<setspec>The set of whole numbers
</set>
<set id=daysofweek ordered>Days of the week
<setitem>Sunday<setitem>Monday<setitem>Tuesday
<setitem>Wednesday<setitem>Thursday<setitem>Friday
<setitem>Saturday
</set>
<set id=colors>Colors
<setitem>Red
<setitem>Green
<setitem>Blue
</set>
<set id=filenamechars>Characters valid in Filenames
 <setref refid=alpha><!-- upper and lowercase alphabetic characters -->
 <setspec setnotation=HyLex>"#"| "$"| "-"
</set>
```

SGML Definition

```
<!ELEMENT  Set              - - (Title?, (setitem | setref | setspec)*) >
<!ATTLIST  Set
    ID              NAME         #REQUIRED
    SetOrder        (ordered | unordered)
                                 unordered
    %Common.Atts.NoID;
>

<!ELEMENT  SetItem          - O (%DataPool;)* >
<!ATTLIST  SetItem
    %Common.Atts;
>

<!ELEMENT  SetRef           - O EMPTY                -- Set Reference -->
<!ATTLIST  SetRef
    ID              NAME         #IMPLIED
    SetID           NAME         #REQUIRED    -- Must be ID of a SET --
    %Common.Atts.Base;

    HyNames         CDATA        #FIXED    '&Common.HyNames;'
    LexModel        CDATA        #FIXED    '&Common.LexModel;'
    RefType         CDATA        #FIXED
                          "SETID SET &Common.RefType;"
>

<!ELEMENT  SetSpec          - O (%DataPool;)*    -- Set Specification -->
<!ATTLIST  SetSpec
    SetNotation     CDATA        #IMPLIED      -- Default is "prose" --
          -- Notation is not required to be an SGML-declared notation --
    %Common.Atts;
>
```

SOA (summary of amendments)

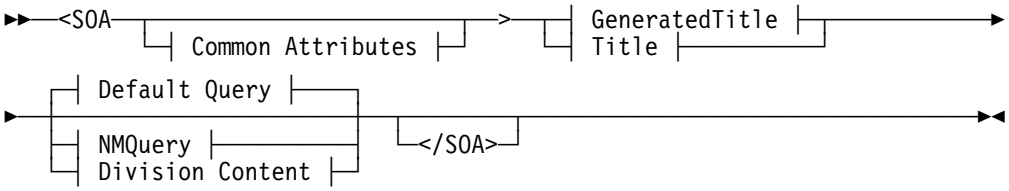
Purpose

SOA is intended to contain information summarizing any changes made to the information since prior versions.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the SOA division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

NMQuery
Default Query

The content of the SOA element can be generated by query. If neither a NMQuery element nor any division content is specified, then the default query is used.

The suggested default query for SOA is to list all elements whose Revision= value refers to an active revision and whose Status= value is not 'unchanged'.

Another useful way to generate the content of the SOA division is to use one or more MarkList elements as the normal division content if you have used marked notes to identify changes in the information.

Division Content

SOA has the same content rules as Div elements.

Usage

The SOA content can be anything, but the marked note elements are particularly handy for generating information about changes. The content of the SOA element can include one or more MarkList elements that generate lists of marked notes where authors have used the notes to mark changes to the document content.

Other queries could be used as well, such as queries that return a list of all elements that have a certain combination of Revision= and Status= attribute values.

If no division content and no NMQuery element is specified, the default SOA query, as defined in the active presentation style, is used to generate the SOA content.

See the MarkNote and MarkList elements for more information on using marked notes to automate the collection of change information.

Examples

```
<frontmatter>
<soa><title>What's Changed Since the Last Release</title>
<p>The following is a list of the major changes made since
the last release of this information:
<MarkList class=changes>
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  SOA              - O (%SpecialDiv;) >
<!ATTLIST  SOA
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

StepNotes

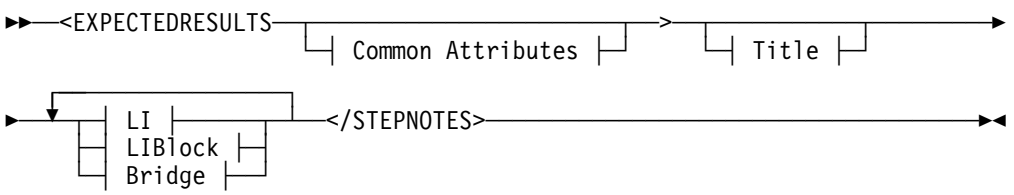
Purpose

Contains an ordered list of notes about the step.

Valid Contexts

ProcStep, following Desc element.

Syntax



Attributes and Contained Elements

Title
Use Title to define an alternate title for the note list. The default English language title for Note List is 'Notes:'.

LI A list item.

LIBlock
Organizes list items within the note list.

Bridge
A Bridge element, normally used to define a transition or linkage between list item blocks or list items.

Usage

Use StepNotes to contain notes about a procedure step, such as special considerations about the step. Do not use StepNotes to convey decision-making information. Rather, use DecisionPoint to identify conditions and actions that may change the path through the procedure.

Examples

```
<procstep><title>Initialize System</title>
<desc>Select the Initialize button to start the initialization
process.
<stepnotes>
<li>You must ensure that you have the correct system level
before starting system initialization.
<li>Values in your configuration profile will affect system
startup.  Be sure you understand the implications of configuration
profiles explained in <xref refid="config-implications">.
</stepnotes>
```

SGML Definition

Style Attribute (element style information or reference)

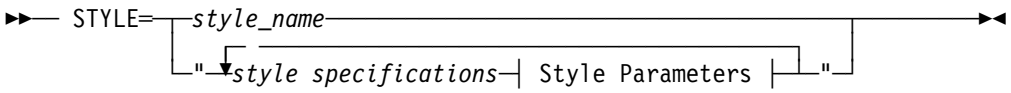
Purpose

The common Style= attribute contains either a reference to a separate style specification for an element or a set of element-specific presentation style definitions when the processing system does not support separate styles for elements.

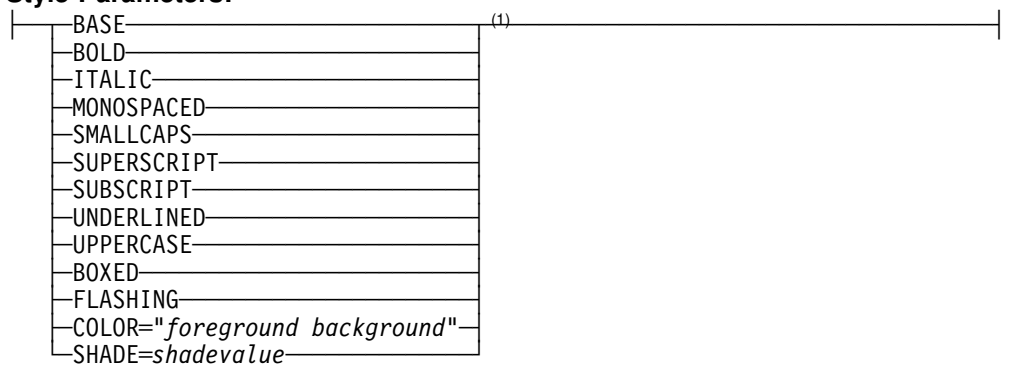
Migration Note

The Style= attribute can contain any DCF BookMaster-defined presentation attributes, such as WIDTH=, SCALE=, TSIZE=, and so on. At some point in the not too distant future, authoring systems will be available that will hide the complexity of associating specific presentation styles with individual elements and the Style= attribute will become a simple reference to a separate element style specification.

Syntax



Style Parameters:



Note:

[1] The style parameters are only valid when used on a PropertyDef element. When specified directly on an element instance, they are ignored.

Attributes and Contained Elements

The following presentation style parameters can be specified on Style= of any PropertyDef element. These keywords are not valid when specified directly for an element instance and will be ignored.

BASE
The element is presented in the base style, regardless of any style it may otherwise inherit.

BOLD
Bold style.

ITALIC
Italic

MONOSPACED
Monospaced font, such as Courier or BookMaster Gothic.

SMALLCAPS
Small caps style.

SUPERSCRIPT
Presented as a superscript.

SUBSCRIPT
Presented as a subscript.

UNDERLINED
The element content is underlined

UPPERCASE
The element content is folded to uppercase.

BOXED
The information is presented boxed. This is intended as a online presentation style for link anchors.

FLASHING
Presented in flashing style. This is intended for online presentation.

COLOR="*foreground background***"**
Presents the information in the specified color. The the foreground and background values are processor-specific color specifications. The background value is optional.

SHADE=*shadevalue*
Indicates that the element should be shaded. The *shadevalue* is a processor-specific shading value, normally a number between zero and 100.

The actual presentation effect of any style parameter is a function of the processing system. For example, ITALIC may be presented as underlined on a system that does not provide italic fonts.

Usage

In its pure form, the Style= attribute is simply a reference to an element style definition in a separate style definition document. In a fully-implemented system, a given Style= attribute specification is associated with a given element using an SGML link process declaration so that the style information is completely separate from the document content. However, because not all processing systems support the full IBM ID Doc style mechanism or the SGML LINK function, the Style= attribute can also be used to contain processor-specific element style specifications in a notation unique to a given processor. When you use this second form, there is no guarantee that a different processor will be able to understand your style definitions.

In order to ensure that your document can be interchanged, it is recommended that style specifications be specified only on PropertyDef elements. This keeps elements and their style specifications separate and enables the specification of different style values for different purposes by applying standard conditional processing mechanisms to the PropertyDef elements (for example, the PROC= attribute).

Migration Note

When migrating from DCF BookMaster to IBM ID Doc, you can specify DCF BookMaster-style presentation attributes as the content of the Style= attribute, as shown below. Given the original DCF BookMaster markup:

```
:figdef id=figdef1 place=inline width=page frame=rules.
```

You would create this IBM ID Doc markup:

```
<PropertyDef ElementType=figure id=figdef1
  style="width=page place=inline frame=rules">
```

Refer to the *BookMaster User&s Guide* for the definition of presentation attributes. Any attribute that controls how an element looks or behaves, and that is not formally defined by IBM ID Doc is a presentation attribute.

Use the Style= attribute on the IBMIDDOC element to define the standard IBM style, for example:

```
<IBMIDDOC style=ibmxagd>
```

Examples

```
<Prolog>
<PropertyDefinitions>
<PropertyDef id=propdef1
      proc=apa
      style="bold italic">
</PropertyDefinitions>
```

SGML Definition

```
    Style           CDATA       #IMPLIED
                                        -- Style specs or reference --
```

Valid Contexts

PropertyDef.

Syntax

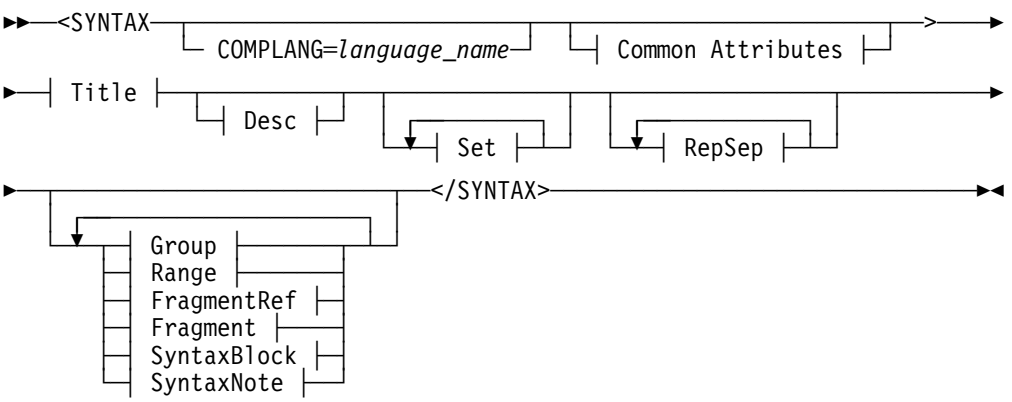
Purpose

Syntax contains the definition of the syntax of a statement in some computer language, such as a command, function call, programming language statement, or other such construct. The markup allows for as complete a definition as possible of the rules of the statement such that automated processes could use the markup to drive generation or validation of real statements. It is also expected that online presentation systems could present more information on request than can be easily or clearly shown graphically.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

COMPLANG=*language_name*
Indicates the computer language this syntax definition is for.

Title
The title of the syntax definition.

Desc
A description of the syntax definition. In the default style, the description is not presented but may be shown by request in an online presentation system, for example.

Set
Defines named sets for use elsewhere in this or other syntax definitions. Sets can be used by reference from ranges and syntactic variables (see “Set (syntactic set)” on page 293 for more information on using sets).

RepSep
Defines any complex repeat separators used by groups within the syntax definition.

Group
Syntax definitions are composed of one or more groups, which then contain and organize the syntactic elements themselves. As a matter of practice, each logical entity within the statement should be contained within its own group. See “Group (syntax grouping)” on page 135 for details on groups.

Range
Range defines a range of values, usually by reference to a set. Ranges are analogous to groups except that their contents is defined algorithmically in terms of set elements, rather than explicitly with syntactic elements. See “Range (syntactic range)” on page 278 for details on ranges.

FragmentRef
A reference to a syntax fragment. The reference is a logical, not a literal reference, so the fragment does not actually have to exist. As for any element, a Linkends= attribute can be used to make the reference literal. Online presentation systems can also use the fragment name to create associative links.

Fragment
Contains a named syntax fragment. Use fragments for constructs that are either re-used, for example in recursive definitions, or for constructs that are too large to fit within the main definition.

Syntax Block
Use syntax blocks to subdivide the elements within a syntax definition into logical divisions and associate titles with them. Unlike groups and fragments, syntax blocks have no syntactic meaning, serving only to organize and further describe the syntax elements.

SyntaxNote
Contains a note about the element that contains it. As for other annotation elements, syntax notes are associated with the element that contains it, which in most cases will be the containing Group element.

Usage

Use syntax definitions to define completely the rules for creating statements in some computer language, for example commands in a command language or programing language expressions. Syntax definitions can also be used to model more abstract constructs, such as the structural relationships between elements in a language.

The default presentation style defined by IDS&T for syntax definitions is as a “railroad track” diagram, but their presentation is not limited to that form. A given definition can be presented in a variety of ways, including BNF or “braces and brackets”. Many of the details of a syntax definition cannot be presented graphically, but in an online presentation system it is expected that those details can be presented at reader request. For example, a variable may refer to a set definition in order to further define the rules for specifying a valid value. The association between the variable and the set cannot be shown graphically, but an online system can present the set definition when the variable is selected by the reader, for example.

Syntax definitions can also be used as the base definition for statements in computer languages. For example, system designers can use the syntax elements to design command interfaces. The markup can also be used to drive interactive systems that help users build valid statements or validate statements given to the system.

Examples

```
<syntax>Month Specification
   <set id=months ordered>Months
     <setitem>JAN
     <setitem>FEB
     <setitem>MAR
     <setitem>DEC
   </set>
   <group>Month Specification
      <kwd>Month_Spec
      <delim start>(
      <range setid=months>Months
      </range>
      <delim end>)
   </group>
</syntax>
```

SGML Definition

```
<!ELEMENT  Syntax           - -
                     (Title, Desc?, Set*, RepSep*,
                      (Group | Range |
                       FragmentRef | Fragment | SyntaxBlock |
                       SyntaxNote)*
                     )
                                                  -- Syntax diagram --
>
<!ATTLIST  Syntax
    CompLang         CDATA        #IMPLIED
       --Defines the computer language to which the definition applies--
```

```
    %Common.Atts;
>
```

SyntaxBlock

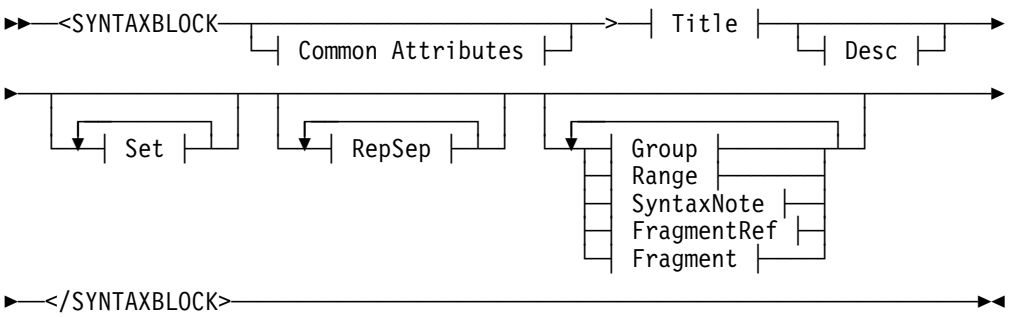
Purpose

Syntax blocks organize syntax definitions into titled subdivisions. Unlike groups and fragments, syntax blocks have no syntactic significance. Syntax blocks are identical to syntax definitions except that they cannot themselves contain syntax blocks.

Valid Contexts

Syntax.

Syntax



Attributes and Contained Elements

Title
The title of the syntax block

Desc
A description of the syntax block. In the default style, the description is not presented but may be shown by request in an online presentation system, for example.

Set
Defines named sets for use elsewhere in this or other syntax definitions. Sets can be used by reference from ranges or syntactic variables (see “Set (syntactic set)” on page 293 for more information on using sets).

RepSep
Defines any complex repeat separators used by groups within the syntax definition.

Group
Syntax definitions are composed of one or more groups, which then contain and organize the syntactic elements themselves. As a matter of practice, each logical entity within the statement should be contained within its own group. See “Group (syntax grouping)” on page 135 for details on groups.

Range
Ranges define a range of values, usually by reference to a set. Ranges are analogous to groups except that their contents are defined algorithmically in terms of set elements, rather than explicitly with syntactic elements. See "Range (syntactic range)" on page 278 for details on ranges.

FragmentRef
A reference to a syntax fragment. The reference is a logical, not a literal reference, so the fragment does not actually have to exist. As for any element, a LinkEnds= attribute can be used to make the reference literal. Online presentation systems can also use the fragment name to create associative links.

Fragment
Contains a named syntax fragment. Use fragments for constructs that are either re-used, for example in recursive definitions, or for constructs that are too large to fit within the main diagram.

Usage

Use syntax blocks to organize the elements of a syntax definition into logical, titled groupings. For example, a single command may have several forms. Syntax blocks allow you to define all the forms in a single syntax definition.

Examples

```
<syntax>Fubar Command
<desc>The fubar command has two forms.
<syntaxblock>Fubar Form A
<desc>Use form A to do A.
<group>Form A
  ⋮
</group>
</syntaxblock>
<syntaxblock>Fubar Form B
<desc>Use form B to do B.
<group>Form B
  ⋮
</group>
</syntaxblock>
</syntax>
```

SGML Definition

```
<!ELEMENT  SyntaxBlock      - O
                     (Title, Desc?, Set*, RepSep*,
                      (Group | Range | FragmentRef | Fragment |
                       SyntaxNote)*
                     )                                -- Syntax block --
>
<!ATTLIST  SyntaxBlock
    %Common.Atts;
>
```

SyntaxNote

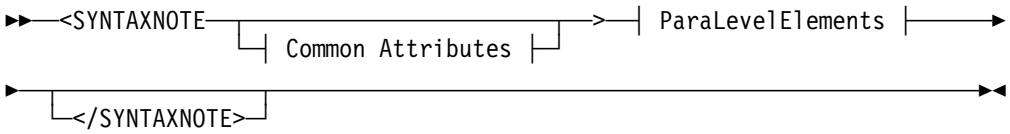
Purpose

Contains a note within a syntax definition group or fragment.

Valid Contexts

Fragment, Group, RepSep, Syntax, SyntaxBlock.

Syntax



Attributes and Contained Elements

Usage

Use SYNTAXNOTE to add notes to your syntax definition to explain aspects of the syntax that cannot be expressed in the syntax markup itself. In the default presentation, the syntax notes are associated with the syntax diagram using numeric callouts and the syntax notes are themselves collected at the end of the presented syntax diagram or syntax block.

Examples

```
<syntax>Attribute Specification
  <group>Attspec
    <kwd>ATTNAME
    <oper>=
    <var>value
    <syntaxnote>
    <p>The results of using this attribute are unpredictable.
    </syntaxnote>
  </group>
</syntax>
```

SGML Definition

```
<!ELEMENT  SyntaxNote       - - (FNBody)               -- Syntax note -->
<!ATTLIST  SyntaxNote
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>
```

SyntaxPhrase (syntax phrase)

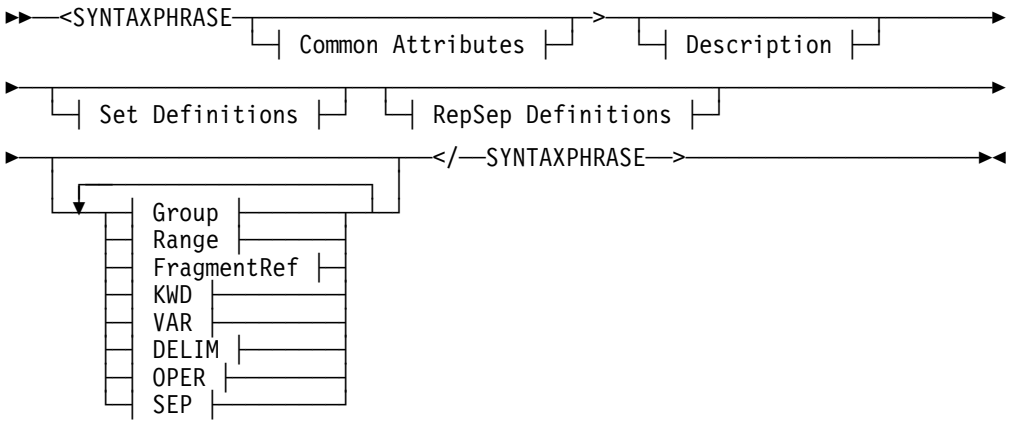
Purpose

The SyntaxPhrase element contains syntax definition elements and is used in the context of the information around it.

Valid Contexts

PhraseLevelElements

Syntax



Usage

The SyntaxPhrase element contains data that is a syntax definition and is used in the context of the text that flows around it.

Migration Note

This element is an extension of the BookMaster PK and PV elements which allows complete syntax diagrams in text. As an example, the PK element might now be marked up as: <syntaxphrase><kwd>*keyword*</kwd></syntaxphrase>.

Use SyntaxPhrase to present syntax fragments outside the context of a complete syntax definition.

Examples

```
To enter this parameter, the syntax
<syntaxphrase><kwd>PARAMETER<delim>=<var>value</syntaxphrase>
must be used.
```

SGML Definition

```
<!ELEMENT  SyntaxPhrase     - - (Desc?, Set*, RepSep*, (%GroupElem;)*)
                                                        -- Syntax phrase --
>
<!ATTLIST  SyntaxPhrase
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "SyntaxPhrase"
>

<!ELEMENT  PK               - - (%DataPool;)*
        -- Parameter keyword == <syntaxphrase><kwd>...</syntaxphrase> --
>
<!ATTLIST  PK
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "SyntaxPhrase"
>

<!ELEMENT  PV               - - (%DataPool;)*
       -- Parameter variable == <syntaxphrase><var>...</syntaxphrase> --
>
<!ATTLIST  PV
    SetID            NAME         #IMPLIED          -- Must be SET ID --
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts.NoRefType;
    RefType          CDATA        #FIXED
                          "SETID SET &Common.RefType;"

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "SyntaxPhrase"
>
```

Table

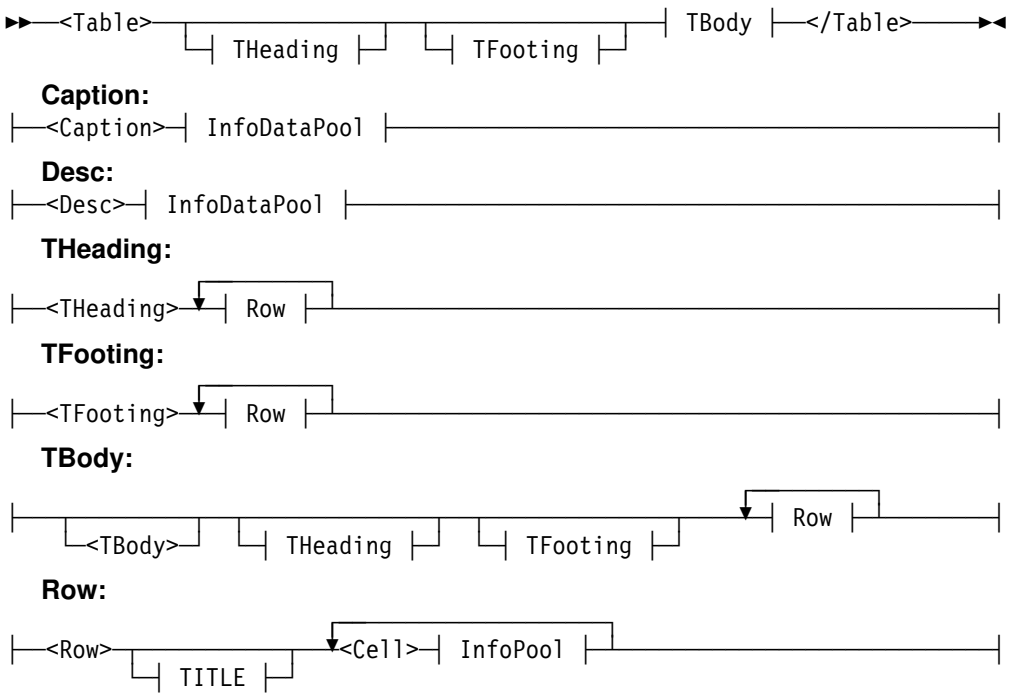
Purpose

TABLE is a simple presentational table which subsumes DIRECT and TABDATA, as well as providing the function of simple BookMaster tables.

Valid Contexts

Figure.

Syntax



Usage

Migration Note

Tables, their definition and markup are a subject which can cause a great deal of heated debate. Virtually every SGML application has its own unique and sophisticated table markup. It is unlikely that any single definition will be invented that is capable of capturing all the intent of all these different table markup systems. It is not the intent of this element to do so. Complex tables is NOT the subject of this element type. “Complex tables” include the database-like tables that BookMaster currently defines. These tables will be supported but not in the base IBM ID Doc implementation; BookMaster tables are a tower in IBM ID Doc. It also does not support very presentation-oriented tables, like CALS tables, which allow great flexibility in spanning rows and columns. CALS tables will also be supported as a tower by IBM ID Doc.

As currently designed, the base IBM ID Doc table support defines a simple n by m format table. It supports the functionality of the DIRECT and TABDATA elements found in BookMaster. It is planned (and supported in the architectural form) that these tables allow limited recursion (table body in a table cell). This allows the construction of complex rows with some spanning capability. This capability is not currently allowed in this specification because of implementation concerns in the prototype.

Like most sequentially processed tabular information, this element defines tables in a row-wise fashion. The number of columns in the table is derived from the number of sibling cells in the first row. The number of cells in all succeeding rows must match the number of cells in the first row. The number of rows is derived from the number of sibling rows found in the table.

The fundamental unit of information in these tables is the cell. There is no provision for logical rearrangement of cells in the markup of the architectural form. This functionality is in the purview of more sophisticated and complex tables. It is possible that the presentation semantics will provide the capability of presenting the information in an order that is different from the logical order defined in the document source.

It is not the intent of this form to replicate or preserve the entry mechanisms used to enter either BookMaster directories or tabbed data. Its objective is to capture the intent of these two table formats. Short references may be defined in the document prolog which allow the use of tabs to enter cells but some editors may lose this markup minimization information.

Headings and footings, information which defines the meaning of the data in the columns, is specified at the start of the table. In addition, row titles may be specified for each row (this is intended to map from the DIRSUB element found in BookMaster).

Usage

```
<!ELEMENT  SimpleTable      - - (THeading?, TFooting?, TBody+) >
<!ATTLIST  SimpleTable
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Table"
>

<!ELEMENT  TBody            O O     (THeading?, TFooting?, TRow+)
                                                -- Table Body --
>
<!ATTLIST  TBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TableBody"
>

<!ELEMENT  THeading         - O (Row+)              -- Table Heading -->
<!ATTLIST  THeading
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TableHeading"
>

<!ELEMENT  TFooting         - O (Row+)              -- Table Footing -->
<!ATTLIST  TFooting
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TableFooting"
>

<!ELEMENT  TRow             - O (Title?, Cell+) >
<!ATTLIST  TRow
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TableRow"
>
```

```
<!ELEMENT  Cell             - O (%InfoPool;)* >
<!ATTLIST  Cell
    N               NMTOKEN      #IMPLIED               -- Cell name --
    %Common.Atts;

    InfoMaster      NAME         #FIXED    "TableCell"
>
```

TableList

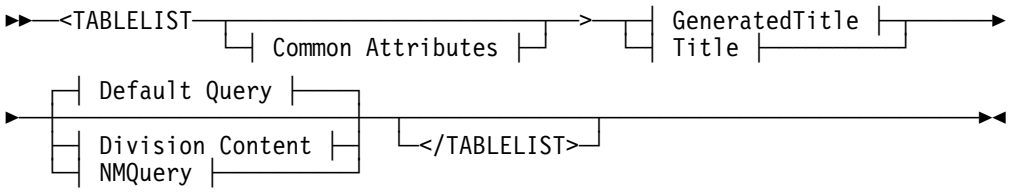
Purpose

TableList contains either an explicit list of figures or a NMQuery element that resolves to a list of figures.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the TableList division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

NMQuery
Division Content

The content of TableList is either a NMQuery element, the content as for other divisions, or nothing.

If you specify no content, then the TableList is defined to represent an application-defined query, where the actual query is defined either in the application implementation or via the application-specific style mechanism.

Default Query

The suggested default query for TableList is to list all Figure elements for which a caption has been specified and that have a Class= value of 'Table'.

Usage

Use TableList to contain or create a list of figures. The simplest use of TableList is as follows:

```
<TableList><GeneratedTitle>
```

This usage causes the presented table list to be created using the default query, which is defined by the processing application. The table list title is also defined by the processing application.

In the case where you need to explicitly define the content of the table list, do so using the normal elements valid within all divisions:

```
<TableList><GeneratedTitle>
<DivBody>
<p>See the following locations for interesting tables.
<ul>
<li><xref refid=table1>
<li><xref refid=table2>
</ul>
</DivBody>
</TableList>
```

Examples

```
<frontmatter>
<TableList><generatedtitle>
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  TableList        - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool;)*)
                                )
>
<!ATTLIST  TableList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>
```

Title (element title)

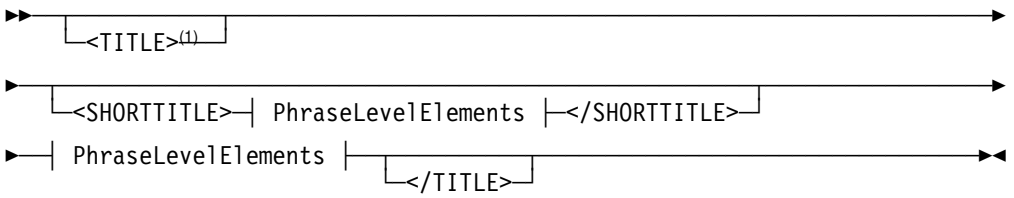
Purpose

The title element contains the title for all elements that can have titles. Title contains an optional short title.

Valid Contexts

Div, P, UL, OL, DL, ParmList, GL, Figure, Annot.

Syntax



Note:
1 The Title start tag is required for all elements in which the title itself is optional, such as for paragraphs and lists.

Attributes and Contained Elements

ShortTitle
Contains an optional short title for the element that contains the Title. In most presentation styles, the short title is used in cross references and other contexts where space is at a premium, such as running heads and feet in print or panel titles online.

The ShortTitle is always optional and you must always specify the ShortTitle start and end tags when you include a short title.

PhraseLevelElements
Characters or any phrase elements.

Usage

The Title element contains the title for any element that takes a title, such as Div. The title can include an optional short title, contained by the ShortTitle element. Both Title and ShortTitle can contain any phrase element as well as normal text.

Examples

In the first example, the Title start tag is implied automatically by the title text:

```
<div>This is the division title
<p>A paragraph
```

Title with short title:

```
<div><shorttitle>The short title</>
This is the long title
<p>A paragraph
```

Title with phrase in the title text:

```
<div>About the <pk>TITLE</pk> Element
<p>Title contains the title.
```

In some writing and presentation styles, you might need or want to put titles on elements other than divisions, such as lists. In these cases, the title element is optional, so you must specify the Title start tag if you include a title:

```
<ul><title>List title
<li>This is a list item
<li>Another list item
</ul>
```

SGML Definition

```
<!ELEMENT  Title           O O (ShortTitle?, TitleBody) >
<!ATTLIST  Title
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Title"
>

<!ELEMENT  ShortTitle       - - (%DataPool;)* >
<!ATTLIST  ShortTitle
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ShortTitle"
>

<!ELEMENT  TitleBody        O O (%DataPool;)* >
<!ATTLIST  TitleBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TitleBody"
>
```

TOC (table of contents)

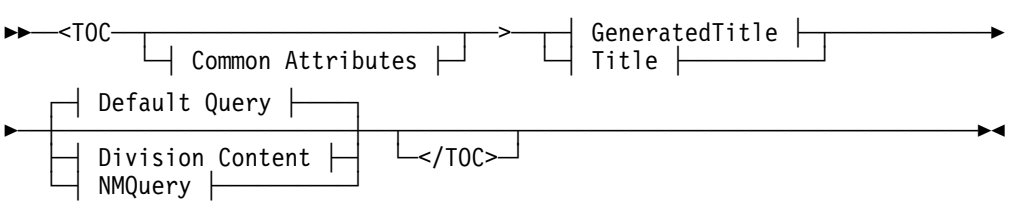
Purpose

TOC contains either an explicit list of divisions or a NMQuery element that resolves to a list of divisions.

Valid Contexts

Frontmatter.

Syntax



Attributes and Contained Elements

GeneratedTitle
Title

You must specify either a GeneratedTitle element or a Title element containing the title for the TOC division.

When you specify GeneratedTitle, the actual title text is defined in the active document style.

If you specify Title, you must specify the Title start tag.

NMQuery
Division Content

The content of TOC is either a NMQuery element, the content as for other divisions, or nothing.

If you specify no content, then the TOC is defined to represent an application-defined query, where the actual query is defined either in the application implementation or via the application-specific style mechanism.

Default Query

The suggested default query for TOC is to list all elements of the InfoMaster architectural form HierarchicalDivision or SpecialDivision.

Usage

Use TOC to contain or create a list of divisions. The simplest use of TOC is as follows:

```
<TOC><GeneratedTitle>
```

This usage causes the presented TOC to be created using the default query, which is defined by the processing application. The TOC title is also defined by the processing application.

In the case where you need to explicitly define the content of the TOC, do so using the normal elements valid within all divisions:

```
<TOC><GeneratedTitle>
<DivBody>
<p>See the following locations:
<ul>
<li><xref refid=div1>
<li><xref refid=div2>
</ul>
</DivBody>
</TOC>
```

Examples

```
<frontmatter>
<TOC><generatedtitle>
  ⋮
</frontmatter>
```

SGML Definition

```
<!ELEMENT  TableList        - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool;)*)
                                )
>
<!ATTLIST  TableList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>
```

UL (unordered list)

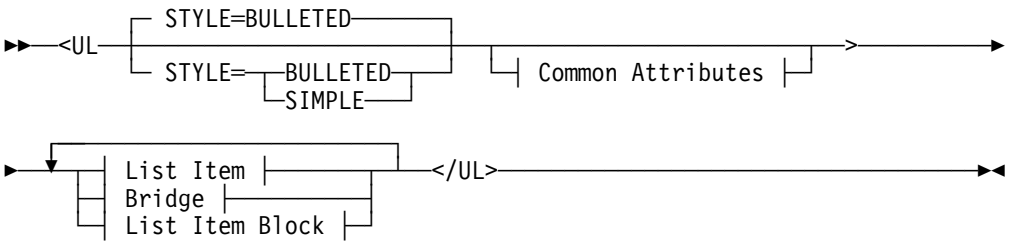
Purpose

Contains an unordered list.

Valid Contexts

ParaLevelElements

Syntax



Attributes and Contained Elements

style=*style choice*
The type of list to be produced. Choices are **bulleted** or **simple**. The default is **bulleted**.

Usage

Paragraphs can contain lists, so you must explicitly end a paragraph before a list if you want the list to be outside the paragraph.

Migration Note

The simple list element from DCF BookMaster has been subsumed into the unordered list element as the only difference is the type of bullet used for the two lists.

Examples

Simple:

```
<ul>
<li>bread
<li>milk
<li>peanut butter
</ul>
```

More complex:

```
<ul>
<li>bread:
<ul>
<li>whole-wheat flour
<li>sandwich-sized slices
</ul>
<li>milk
<li>peanut butter
</ul>
```

SGML Definition

```
<!ELEMENT  UL              - - (Title?, (LI | LIBlock | Bridge)+)
                                                  -- Unordered List --
>
<!ATTLIST  UL
    %Common.Atts

    InfoMaster       NAME         #FIXED    "SinglePartList"
>
```

Var (syntactic variable)

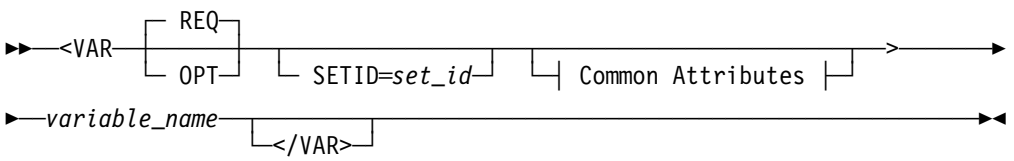
Purpose

Defines a syntactic variable within a syntax definition group, fragment, or SynPhrase.

Valid Contexts

Group, RepSep, SyntaxPhrase.

Syntax



Attributes and Contained Elements

SETID=*set_id*
The ID of a SET definition that defines the set of characters valid for the variable value.

REQ | **OPT**
Indicates whether or not the variable is optional. REQ (required) is the default.

variable_name
The name of the syntactic variable. The name is presented in the syntax diagram.

Usage

Use Var to define syntactic variables within a syntax definition. You can refer to a set definition in order to define the set of characters usable in specifying the variable value.

Examples

```
<syntax>Attribute Specification
  <group>Attspec
    <sep>&ssbl;
    <kwd>ATTNAME
    <oper>=
    <var>value
  </group>
</syntax>
```

SGML Definition

```
<!ELEMENT  Var              - O (%DataPool;)*      -- Syntax variable -->
<!ATTLIST  Var
    SetID            NAME         #IMPLIED          -- Must be SET ID --
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts.NoRefType;

    RefType          CDATA        #FIXED
                          "SETID SET &Common.RefType;"
>
```

Warning (warning notice)

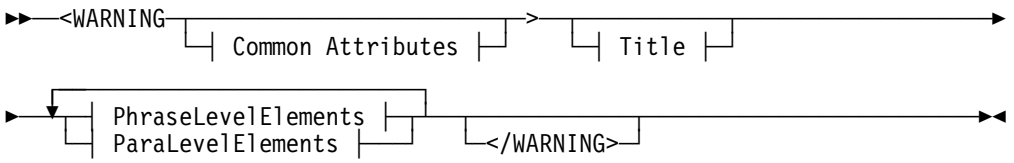
Purpose

Contains a warning notice.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

Title
In the default style, the generated English language title for Warning is 'Warning:'. When specified, the Title replaces any generated title.

PhraseLevelElements
ParaLevelElements
Warning has the same content model as paragraphs, except that Warning can contain paragraphs

Usage

Use Warning to create a warning notice, consisting of one or more paragraphs or other paragraph-level elements. Warnings are normally used to warn about actions that may cause damage to data.

Examples

Simple example:

```
<warning>Turning off your computer without first shutting down
may result in the loss of data.
</warning>
```

Example with title:

```
<Warning><title>Warning title</title>
Warning content
```

SGML Definition

```
<!ELEMENT  Warning          - O (Title?, NoteBody) >
<!ATTLIST  Warning
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>
```

Web (hyperlink web)

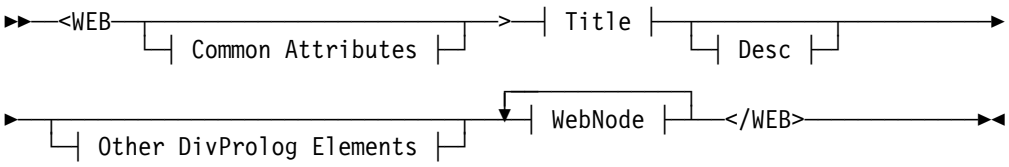
Purpose

Web contains one or more web nodes, which then define hyperlinks to either other nodes or content elements.

Valid Contexts

WebSet.

Syntax



Attributes and Contained Elements

Title
The title of the web.

Desc
A description of the purpose of the web. Although the Desc element is optional, it is strongly recommended that all webs contain a description.

Other DivProlog Elements
Any other DivProlog elements can be used within Web. For example, you can use PropertyDefinitions to define anchor roles with AnchorRoleDef.

Usage

Use Web to create an organizing set of hyperlinks for the purpose of providing non-sequential access to information using hypertext presentation and access methods. A web is made up of one or more WebNode elements. WebNode elements define links to other nodes or to content information and associate descriptive information with those links. Webs can be used to create arbitrary organizations of hyperlinks for use with online presentation systems. Webs are primarilly intended for use with online presentation systems, but it is possible to define print representations of webs. Webs, by definition, have no defined order of presentation except for the first WebNode in each web, which is always the primary entry point to the web.

WebNode IDs are always scoped at the Web level. This means that WebNode IDs only need to be unique within a given Web.

Examples

```
<IBMIDDOC>
<Body>

  ⋮
<WebSet>
<Web id=prod-understanding-web>Understanding &prodname;
 <desc>This web organizes access to the information
 you need in order to understand how to make the
 best use of &prodname;.
 <p>This is a simple web in which all LINKENDS= specified links
```

```
are to other WebNodes or division elements in the main document body.
</desc>
<WebNode id=mainnav
         linkends="aboutA aboutB aboutC">
Main Topics for Understanding &prodname;
<p>
</WebNode>
<WebNode id=aboutA
         Linkends="diva1 diva2 diva3">
Understanding Aspect A of &prodname;
<p>These topics will help you understand
the A aspect of &procname;
</WebNode>
<WebNode id=aboutB
         linkends="divb3 divb4 divb5">
Understanding Aspect B of &prodname;
<p>Aspect B of &prodname; is primarily concerned
with the framitizing and deframitizing task.
These topics will help you better understand
framitizing.
</WebNode>
</Web>
<Web id=prod-maintasks-web>Using &prodname;:  Main User Tasks
 <desc>This web organizes access to the information you
 need in order to perform the main user tasks involved with
 using &prodname;.
 <WebNode id=mainnav>Main Tasks for &prodname;
  ⋮
</web>
</WebSet>
</IBMIDDOC>
```

SGML Definition

```
<!ELEMENT  Web              - O (DivProlog, WebNode+)
                                              -- Contains web nodes --
>
<!ATTLIST  Web
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>
```

WebNode (hyperlink web node)

Purpose

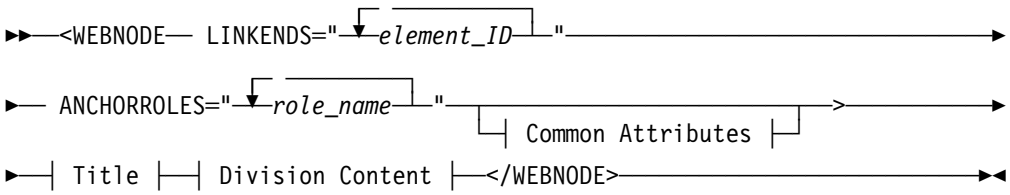
Web nodes define hyperlinks to either other nodes or content elements within webs and associate descriptive information with those links.

WebNode (hyperlink web node)

Valid Contexts

Web.

Syntax



Attributes and Contained Elements

LINKENDS=*element_IDs*
The common LinkEnds= attribute (see -- Heading 'LINKEND' unknown -- for details).

ANCHORROLES=*role_names*
The common AnchorRoles= attribute (see -- Heading 'ANCHROL' unknown -- for details).

Title
The title of the webnode.

Division Content
WebNodes have the same content rules as Div elements.

Usage

Use WebNode elements to define links to other nodes or to content information and associate descriptive information with those links. Webs can be used to create arbitrary organizations of hyperlinks for use with online presentation systems. The WebNode content should explain the purpose of the node and the information to which it links.

The first WebNode within a Web is defined to be the primary node or entry point node for the web. However, the details of how webs are presented and accessed is a function of the online presentation system you are using. A given system is free to define other rules for node presentation and access.

Use the common LinkEnds= attribute on the WebNode element to define the hyperlink anchors for the node. You can use the common AnchorRoles= attribute to assign specific anchor roles to the anchors, if desired. There is no special significance attached to the order the anchors are specified in LinkEnds=. You can define anchor roles specific to your information using the AnchorRoleDef element within PropertyDefinitions, either at the Web level or within a given WebNode. See “AnchorRoleDef (hyperlink anchor role defininition)” on page 45 for more information on defining and using anchor roles.

Examples

```
<IBMIDDOC>
<Body>

  ⋮
<WebSet>
<Web id=prod-understanding-web>Understanding &prodname;
 <desc>This web organizes access to the information
 you need in order to understand how to make the
 best use of &prodname;.
 <p>This is a simple web in which all LINKENDS= specified links
 are to other WebNodes or division elements in the main document body.
 </desc>
 <WebNode id=mainnav
          linkends="aboutA aboutB aboutC">
 Main Topics for Understanding &prodname;
 <p>
 </WebNode>
 <WebNode id=aboutA
          Linkends="diva1 diva2 diva3">
 Understanding Aspect A of &prodname;
 <p>These topics will help you understand
 the A aspect of &procname;
 </WebNode>
 <WebNode id=aboutB
          linkends="divb3 divb4 divb5">
 Understanding Aspect B of &prodname;
 <p>Aspect B of &prodname; is primarily concerned
 with the framitizing and deframitizing task.
 These topics will help you better understand
 framitizing.
 </WebNode>
</Web>
<Web id=prod-maintasks-web>Using &prodname;:  Main User Tasks
 <desc>This web organizes access to the information you
 need in order to perform the main user tasks involved with
 using &prodname;.
 <WebNode id=mainnav>Main Tasks for &prodname;
  ⋮
</web>
</WebSet>
</IBMIDDOC>
```

SGML Definition

```
<!ELEMENT  WebNode          - O (%Division;)   -- Contains a web node -->
<!ATTLIST  WebNode
    ID              NAME         #REQUIRED
    %Common.Atts.NoID;

    InfoMaster      NAME         #FIXED    "ContainingDivision"
>
```

WebSet (set of hyperlink webs)

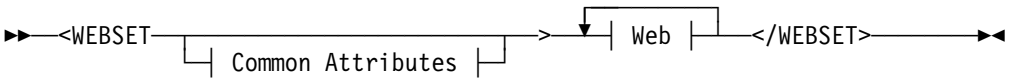
Purpose

WebSet contains one or more hyperlink webs within IBMIDDOC.

Valid Contexts

IBMIDDOC.

Syntax



Attributes and Contained Elements

Web
One or more Web elements.

Usage

Use WebSet to contain one or more hyperlink webs. A web is made up of one or more WebNode elements. WebNode elements define links to other nodes or to content information and associate descriptive information with those links. Webs can be used to create arbitrary organizations of hyperlinks for use with online presentation systems. Webs are primarily intended for use with online presentation systems, but it is possible to define print representations of webs. Webs, by definition, have no defined order of presentation except for the first WebNode in each web, which is always the primary entry point to the web. See “Web (hyperlink web)” on page 319 for more information.

Examples

```
<IBMIDDOC>
<Body>

  ⋮
<WebSet>
<Web id=prod-understanding-web>Understanding &prodname;
 <desc>This web organizes access to the information
 you need in order to understand how to make the
 best use of &prodname;.
 <p>This is a simple web in which all LINKENDS= specified links
 are to other WebNodes or division elements in the main document body.
 </desc>
 <WebNode id=mainnav
          linkends="aboutA aboutB aboutC">
 Main Topics for Understanding &prodname;
 <p>
 </WebNode>
 <WebNode id=aboutA
          Linkends="diva1 diva2 diva3">
```

```
 Understanding Aspect A of &prodname;
 <p>These topics will help you understand
 the A aspect of &procname;
 </WebNode>
 <WebNode id=aboutB
          linkends="divb3 divb4 divb5">
 Understanding Aspect B of &prodname;
 <p>Aspect B of &prodname; is primarily concerned
 with the framitizing and deframitizing task.
 These topics will help you better understand
 framitizing.
 </WebNode>
</Web>
<Web id=prod-maintasks-web>Using &prodname;:  Main User Tasks
 <desc>This web organizes access to the information you
 need in order to perform the main user tasks involved with
 using &prodname;.
 <WebNode id=mainnav>Main Tasks for &prodname;
  ⋮
</WebSet>
</IBMIDDOC>
```

SGML Definition

```
<!ELEMENT  WebSet           - O (Web+)     -- Contains hyperlink webs -->
<!ATTLIST  WebSet
    %Common.Atts;
>
```

Xmp (example of computer input or output)

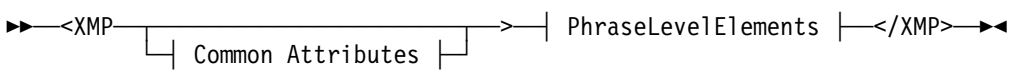
Purpose

Contains an example of computer input or output, such as code samples or listings. Within Xmp, significant record ends are preserved and presented.

Valid Contexts

ParaLevelElements.

Syntax



Attributes and Contained Elements

PhraseLevelElements

Xmp can contain any phrase-level elements, including Litdata. You can use Litdata within Xmp to include data such as code samples without having to worry about constructs in the data interacting with the SGML parsing.

Usage

Use Xmp to contain examples of computer input or output, such as code samples or listings. In the default style, Xmp data is presented in a monospaced typeface.

Examples

```
<xmp>
<litdata name=cprog1>
</xmp>
```

SGML Definition

```
<!ELEMENT  XMP              - O (%DataPool;)*
                                 -- computer input and output example --
>
<!ATTLIST  XMP
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    Notation         (LineSpecific)
                                  #FIXED    "LineSpecific"
    ApplicationClass NAME         #FIXED    "ComputerExample"
>
```

XPH (example phrase)

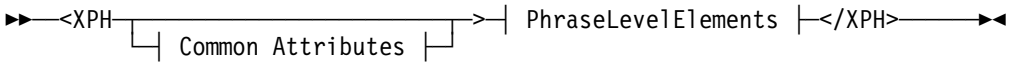
Purpose

The XPH element contains data that is a computer input or output and is used in the context of the text that flows around it.

Valid Contexts

PhraseLevelElements

Syntax



Usage

The XPH element contains data that is a computer input or output and is used in the context of the text that flows around it. The content of the element may be any data or other elements from PhraseLevelElements. In the default style, XPH is presented in the same typeface as is used for XMP elements.

Examples

```
The system will respond with a <XPH>READY</XPH> message.
```

SGML Definition

```
<!ELEMENT  XPH              - - (%DataPool;)*
                            -- Computer input or output example phrase --
>
<!ATTLIST  XPH
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "ExamplePhrase"
>
```

XRef (cross reference)

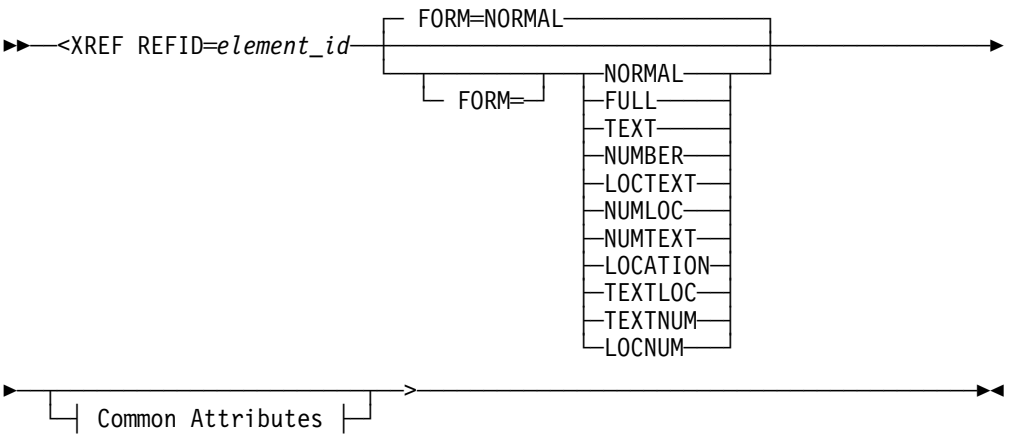
Purpose

XRef defines a reference to another element and generates the reference text (or other link indicator) automatically. The element referred to can be in the same document or in another document. XRef can point to another element indirectly by refering to a LinkDesc element.

Valid Contexts

MultiMediaObject and wherever phrase-level elements are valid.

Syntax



Attributes and Contained Elements

REFID=*element_id*

The ID= value of the element being linked to. The linked element may be any element that takes an ID= attribute.

Note: RefID= is the HyTime LinkEnd= attribute.

FORM=*formtype*

Determines which of the three possible elements of the generated cross reference are to be used: text, number, or location. The generated text is either the element title, for elements that take a title element (for example, divisions), or the XRefText= value specified on the element linked to. The generated location is either a page number for printed output or an online location specification for online systems (for example, a topic ID in BookManager). The number is for enumerated elements only (for example, ordered list items). The specific choices are:

NORMAL
Whatever the active presentation style defines a normal cross reference to be.

FULL
A full reference including all available information about the target element.

TEXT
Just the text of the target element, typically its title.

NUMBER
Just the number of the target element, if it is enumerated (the meaning of NUMBER for an unenumerated element is undefined).

LOCATION
The location of an element, such as its page number or panel ID. In some presentations, there may not be a meaningful locator, in which case the presentation style may do the best it can, such as presenting the title of the nearest containing element.

NUMLOC
The number of an enumerated element followed by its location.

NUMTEXT
The number of an enumerated element followed by its text.

LOCTEXT
The location of an element followed by its text.

TEXTLOC
The text of an element followed by its location.

TEXTNUM
The text of an element followed by its number.

LOCNUM
The location of an element followed by its number.

Usage

XRef defines a link to another element and automatically generates the link indicator. The link indicator is normally the title of the element linked to along with some locator, such as a page or panel reference, or, when the element does not have a title, just a locator. When the element linked to has an XRefText= attribute specified, the XRefText= value is used as the link indicator. For example, a cross reference to a paragraph that does not have XRefText= coded would generate just a page number.

The style of the generated cross reference is determined by the active document style.

You can create a cross reference indirectly by using a LinkDesc element to define the target of the cross reference. For example, you may have already defined a LinkDesc to something for use by the Link element because it will be linked to many times. You can use this same LinkDesc for cross references to the element.

Examples

Refering directly to another element:

```
<p>See <xref refid=AboutXRef> for more information.
  ⋮
<hd id=AboutXref>XRef Explained
```

SGML Definition

```
<!ELEMENT  XRef             - O EMPTY      -- Cross Reference Element -->
<!ATTLIST  XRef
    ID               NAME         #IMPLIED
    RefID            NAME         #REQUIRED
    Form             (normal | full | text | number | location |
                      textnum | locnum | textloc |
                      numtext | numloc | loctext)
                               normal
    %Common.Atts.Base;

    InfoMaster       NAME         #FIXED    "CrossReference"
    -- HyTime attributes: --
    HyTime           NAME         #FIXED    "ilink"
    reftype          CDATA        #FIXED
                          "#ALL #ANY &Common.RefType;"
                        -- No REFLEVEL restrictions (see 10744 6.5.3) --
    refrange         CDATA        #FIXED    "#ALL I"   -- indirection --
    LexModel         CDATA        #FIXED
                          "RefID IDREFS &Common.LexModel;"
    HyNames          CDATA        #FIXED
                          "linkend RefID &Common.HyNames;"
>
```

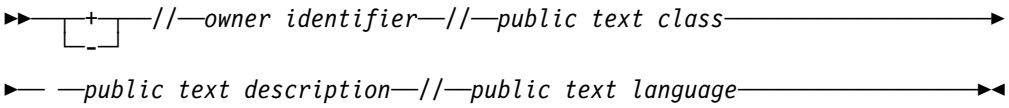
Part 3. Appendixes

© Copyright IBM Corp. 1992, 1993

Appendix A. Proposed IBM Standard for Formal Public Identifiers

One of the SGML mechanisms that InfoMaster depends on is Formal Public Identifiers (FPIs). This is a system-independent naming format that is defined in the standards ISO 8879 and ISO 9070. As part of this syntax, the standard provides for the use of a structure called an owner identifier. There are two forms of interest, registered and unregistered. There is a plan to have a registry of owner identifiers (this plan is codified in ISO 9070). In addition, within the 9070 standard there is provision for a registered identifier based on an ISBN number. Either an ID based on the 9070 registration or ISBN registration may be used. IBM has a publisher prefix, 0-933186, which should be used for all formal public identifiers for IBM-owned objects. Following the owner identifier, there is a field called the public text description. The remainder of this section discusses a convention for using this public text description field to identify IBM-owned entities. This proposal uses the Public Text Description field of the FPI for use as part of the IBM ID Doc application. This should become a standard within IDS&T if not with others within the corporation. Our customers may wish to use a definition like that defined here for their own data.

The basic format of a formal public identifier is:



A typical FPI for an IBM document, using the conventions in this standard, would look like this:

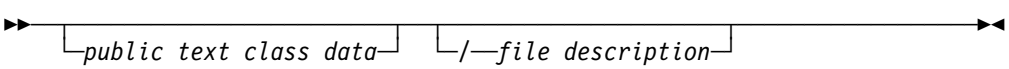
```
+//ISBN 0-933186::IBM//DOCUMENT PUB SC31-1234-00//EN
```

The '+//ISBN ...' is the registered owner identifier, in this case indicating that the owner is IBM using the ISBN registration. Notice that **0-933186** represents the ISBN registration identifier that has been assigned to IBM. The alternative ISO 9070 format looks like this:

```
+//ISO/IEC 9070/RA::nnnn::US::IBM//DOCUMENT PUB SC31-1234-00//EN
```

The '+//ISO/IEC 9070/RA::...' is the registered owner identifier, in this case indicating that the owner is IBM using the ISO 9070 registration. Notice that **nnnn** represents the number assigned by the registration authority, this number has not yet been assigned to IBM. The keyword **DOCUMENT** defines the public text class. The list of possible values for this field are defined in ISO 8879. It is also reiterated in the list below. The keyword **PUB** is the start of the public text description field, which this standard covers. The keyword **PUB** in this context indicates an IBM publication and is followed by the standard IBM publication number. The '//EN' is the public text language identifier, in this case, English. The list of language codes is defined in ISO 639.

The public text description field has the following format:



© Copyright IBM Corp. 1992, 1993

The following list defines the *public text class data* field format. Notice that its value is keyed on the public text class. Unless otherwise noted, the data in the field identifies the entity itself; in some cases it may refer to either a containing context or another entity to which the defined information is related. As used in these definitions, the term “context” means the document library or collection to which the entity applies or which owns the entity.

Public Text Class **Public Text Class Data format**

CAPACITY

The following field identifies the DTD to which this capacity set entity applies.

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]

A corporate standard should be written for all IBM internal DTDs which are distributed throughout the company. The CAPACITY file may be used to maintain the CAPACITY used by these DTDs and is referenced from SGML declaration files which apply to the DTD.

CHARSET [CP nnnnn[-n]]

where

nnnnn[-n] indicates a standard IBM codepage, if specified. If not specified, the character set is assumed to not be an IBM-defined codepage.

Notice that if another standard is used for character encoding, such as an ANSI or ISO standard, the FPI for their standards, as they define them, should be used.

DOCUMENT

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]
IBM publication PUB zznn-nnnn[-nn]
IBM collection LIB zznn-nnnn[-nn]
ISBN notation ISBN n-nn-nnnnnn-n

DTD

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]

A corporate standard should be written for all IBM internal DTDs which are distributed throughout the company.

Notice that if a standard DTD is used from another institution, like ANSI or ISO, the FPI for their standards, as they define them, should be used.

ELEMENTS

If the declared elements are intended for general use throughout IBM, the entity would be identified by the corpo-

rate standard identifier of the DTD to which the declarations are related.

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n][::decl descriptor]

If elements are defined for a particular document, library or other collection, the identifier identifies the owning context.

IBM publication PUB zznn-nnnn[-nn][::decl descriptor]

IBM collection LIB zznn-nnnn[-nn][::decl descriptor]

ISBN notation ISBN n-nn-nnnnnn-n[::decl descriptor]

where

decl descriptor
is a unique identifier within the scope of the owning context.

ENTITIES

When an entity set is distributed throughout the company, a corporate standard should be written for it and it should be identified using this field:

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]

When an entity set is used to hold the entity declarations for a particular document, library or collection, the identifier for that context should be used to identify the entity:

IBM publication PUB zznn-nnnn[-nn][::decl descriptor]

IBM collection LIB zznn-nnnn[-nn][::decl descriptor]

ISBN notation ISBN n-nn-nnnnnn-n[::decl descriptor]

where

decl descriptor
is a unique identifier within the scope of the owning context.

Notice that if a standard entity set is used from another institution, like American Mathematical Society or ISO, the FPI for their standards, as they define them should be used.

LPD

When an link process declaration set is distributed throughout the company, a corporate standard should be written for it and it should be identified using this field:

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]

When an LPD is used to hold the declarations for a particular document, library or collection, the identifier for that context should be used to identify the entity:

IBM publication PUB zznn-nnnn[-nn][::decl descriptor]

IBM collection LIB zznn-nnnn[-nn][::decl descriptor]
ISBN notation ISBN n-nn-nnnnnn-n[::decl descriptor]

where

decl descriptor
is a unique identifier within the scope of the owning context.

NONSGML

The following fields identify the owning context for the non-SGML entity.

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n][::data descriptor]
IBM publication PUB zznn-nnnn[-nn][::data descriptor]
IBM collection LIB zznn-nnnn[-nn][::data descriptor]
ISBN notation ISBN n-nn-nnnnnn-n[::data descriptor]

where

data descriptor
is a unique identifier within the scope of the owning context.

NOTATION

The following field identifies the document which defines the notation referred to by this notation identifier.

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]

Notice that if a notation that is defined by standards body, like ANSI or ISO, the FPI for their standards, as they define them should be used.

SHORTREF

When an short reference declaration set is distributed throughout the company, a corporate standard should be written for it and it should be identified using this field:
Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]
When an short reference set is used to hold the declarations for a particular document, library or collection, the identifier for that context should be used to identify the entity:
IBM publication PUB zznn-nnnn[-nn]
IBM collection LIB zznn-nnnn[-nn]
ISBN notation ISBN n-nn-nnnnnn-n

SUBDOC

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]
IBM publication PUB zznn-nnnn[-nn]
ISBN notation ISBN n-nn-nnnnnn-n

SYNTAX

The following field identifies the DTD to which this syntax definition entity applies.

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n]

A corporate standard should be written for all IBM internal DTDs which are distributed throughout the company. The SYNTAX file may be used to maintain the SYNTAX used by these DTDs and is referenced from SGML declaration files which apply to the DTD.

When an syntax definition only applies to a particular document, library or collection, the identifier for that context should be used to identify the entity:

IBM publication PUB zznn-nnnn[-nn]
IBM collection LIB zznn-nnnn[-nn]
ISBN notation ISBN n-nn-nnnnnn-n

TEXT

The following fields identify the owning context for the SGML text entity.

Corporate Standard STD ppp n-nnnn-nnn [nnnn-n][::text descriptor]
IBM publication PUB zznn-nnnn[-nn][::text descriptor]
IBM collection LIB zznn-nnnn[-nn][::text descriptor]
ISBN notation ISBN n-nn-nnnnnn-n[::text descriptor]

where

text descriptor
is a unique identifier within the scope of the owning context.

The text *file description* has the following format:

```
data description[/qualified name[/owner]]
```

where

data description
is any author-defined description that identifies the content of the entity being described.

In the absense of a *public text class data*, this field must serve to make the formal public identifier unique within the scope of its use.

qualified name
This is a name which uniquely defines the author. It may be an RSCS address, an Internet address, an employee serial number. The key is that it must be unique across the corporation.

owner
identifies the owner of the data. This need not be unique; it could be the name of the author.

During processing, FPIs are normalized using the same normalization rules as those used for tokenizing attribute value literals:

1. Ignore RS
2. Replace RE and SEPCHAR with SPACE
3. Replace a sequence of SPACE characters with a single SPACE and
4. Ignore leading and trailing SPACE characters.

Appendix B. IBM ID Doc Naming Conventions

IBM ID Doc uses the following naming conventions for the names of all constructs defined by IBM ID Doc:

- Bicapitalization is used for all element, attribute, and entity names. SGML keywords are shown in upper case.
- In most cases, words used in element and attribute names are completely spelled out except in the following cases:
  - Where element names have been carried over from the DCF BookMaster language, such as OL, UL and XMP.
  - Where element names would become unnecessarily long. In these cases, the following abbreviations have been used:

    Bib Bibliography, e.g., BibRecord.

    Def Definition, e.g., ClassDef.

    Desc Description, e.g., Desc element.

    Div Division, e.g., Div and DivIntro elements.

    Info Information, e.g., InfoModule element.

    Lib Library, e.g., LibRecord.

    Mod Module, e.g., ModularInfoDef element.

    Ref Reference, e.g., XRef.

    Syn Syntax, e.g., SynPhrase.

Capitalization may differ in some contexts. For example, coding examples may use different capitalization than the DTD fragments or the text. IBM ID Doc is defined such that the case of names is not significant for tags and attributes. Name case is significant for entity names.

© Copyright IBM Corp. 1992, 1993

Appendix C. Predefined HyQ Query Functions for IBM ID Doc

The following are the predefined HyQ query functions defined for IBM ID Doc. These queries represent the suggested default queries for the various QueryLink form elements (TOC, Figlist, GL, etc.). While these queries should be directly usable by any system that supports the HyQ language, it is not necessary to support or implement HyQ. These query specifications are provided primarily as an aid to IBM ID Doc system implementors; you are free to use any query notation or method desired to actually implement these queries. For example, the default queries for tables of contents can be represented by existing TOC generation facilities in composition or online presentation systems without the need to literally create the TOC by query[8]

The domain for all the default queries is the current document.

Note that it is up to the presentation system to generate the presentation of the results of any given query. For example, in the case of a TOC, the presentation system would probably retrieve the Title content for each element and present it indented relative to the nesting level of its parent division. This sort of processing is purely a matter of presentation style and therefore is not reflected in the base IBM ID Doc queries.

All_BibRecords()

Purpose

Returns a list of all BibRecord and IBMBibRecord elements in the query domain, sorted by either the text of the document title or author's name.

HyQ Query Definition

```
<HyQ fn="All_BibRecords">
 <!-- TBD -->
</HyQ>
```

Arguments

None.

Returned Data

Returns a node list containing the address of the located BibRecord and IBMBibRecord elements, sorted alphabetically.

[8] In other words, any processing function that generates a list of things can be represented by a query, specifically by a HyQ query, but it is not necessary to actually implement list generation using a query mechanism.

© Copyright IBM Corp. 1992, 1993

Examples

```
<gl>
 <nmquery useq="All_BibRecords">
</gl>
```

All_Divisions_To_Level(level)

Purpose

Returns a list of all elements of architectural form HierarchicalDivision or SpecialDivision whose level of nesting is less than or equal to the specified value.

Note that it is up to the presentation system to generate the presentation of the results of the query. For example, in the case of a TOC, the presentation system would probably retrieve the Title content for each element and present it indented relative to the nesting level of its parent division. This sort of processing is purely a matter of presentation style and therefore is not reflected in the base IBM ID Doc query.

HyQ Query Definition

```
<HyQ fn="All_Divisions_To_Level">
 <!-- TBD -->
</HyQ>
```

Arguments

level
The nesting level used to limit the query. The nesting level is defined as the number of HierarchicalDivision parents for a given HierarchicalDivision element.

Returned Data

Returns a node list containing the address of the located HierarchicalDivision or SpecialDivision elements.

Examples

```
<toc><generatedtitle>
 <nmquery useq="All_Divisions_To_Level" parms="2">
</toc>
```

All_Figures()

Purpose

Returns a list of all Figure elements in the query domain except those that have a class of Table.

HyQ Query Definition

```
<HyQ fn="All_Figures">
 <!-- TBD -->
</HyQ>
```

Arguments

None.

Returned Data

Returns a node list containing the address of the located Figure elements.

Usage

Note that it is up to the presentation system to define the presentation order for the located Figure elements if the order of occurrence is not the same as the order of presentation or numbering.

Examples

```
<figurelist><generatedtitle>
 <nmquery useq="All_Figures">
</figurelist>
```

All_Figures_of_Class_X()

Purpose

Returns a list of all Figure elements in the query domain except those that are of the specified class.

HyQ Query Definition

```
<HyQ fn="All_Figures_of_Class_X">
 <!-- TBD -->
</HyQ>
```

Arguments

classname
The Class= value to match.

Returned Data

Returns a node list containing the address of the located Figure elements.

Usage

Note that it is up to the presentation system to define the presentation order for the located Figure elements if the order of occurrence is not the same as the order of presentation or numbering.

Examples

```
<figurelist><generatedtitle>
 <nmquery useq="All_Figures_of_Class_X" args="example">
</figurelist>
```

All_GLEntries()

Purpose

Returns a list of all glossary list entries in the query domain, sorted by either the text of the contained DefinedTerm or the SortKey= value of the GLEntry element.

HyQ Query Definition

```
<HyQ fn="All_GLEntries">
 <!-- TBD -->
</HyQ>
```

Arguments

None.

Returned Data

Returns a node list containing the address of the located GLEntry elements, sorted alphabetically.

Examples

```
<gl>
 <nmquery useq="All_GLEntries">
</gl>
```

All_LibRecords()

Purpose

Returns a list of all LibRecord and IBMLibRecord elements in the query domain, sorted by either the text of the document title or author's name.

HyQ Query Definition

```
<HyQ fn="All_LibRecords">
 <!-- TBD -->
</HyQ>
```

Arguments

None.

Returned Data

Returns a node list containing the address of the located LibRecord and IBMLibRecord elements, sorted alphabetically.

Examples

```
<gl>
 <nmquery useq="All_LibRecords">
</gl>
```

All_Safety_Notices()

Purpose

Returns a list of all Caution, Warning, and Danger elements in the query domain.

HyQ Query Definition

```
<HyQ fn="All_Safety_Notices">
 <!-- TBD -->
</HyQ>
```

Arguments

None.

Returned Data

Returns a node list containing the address of the located Caution, Warning, and Danger elements.

Examples

```
<tablelist><generatedtitle>
 <nmquery useq="All_Tables">
</tablelist>
```

All_Tables()

Purpose

Returns a list of all Figure elements in the query domain except that have a class of Table.

HyQ Query Definition

```
<HyQ fn="All_Tables">
 <!-- TBD -->
</HyQ>
```

Arguments

None.

Returned Data

Returns a node list containing the address of the located Figure elements.

Usage

Note that it is up to the presentation system to define the presentation order for the located Figure elements if the order of occurrence is not the same as the order of presentation or numbering.

Examples

```
<tablelist><generatedtitle>
 <nmquery useq="All_Tables">
</tablelist>
```

Appendix D. IBM ID Doc Document Type Definition

This section contains the IBM ID Doc document type definition (DTD).

```
<!--====================================================================
 Copyright:
            (c) International Business Machines Corporation 1992, 1994
            All Rights Reserved.

Classification:  IBM Internal Use Only

       Version:  1.0

          File:  IBMIDDOC DTD
 Formal Public
   Identifiers:
                Most recent version:
                +//ISBN 0-933186::IBM//DTD IBMIDDoc//EN

                This version:
                +//ISBN 0-933186::IBM//DTD IBMIDDoc:1.0//EN

      Function:  Markup declarations for the IBMIDDOC document type

Change History:   4 Aug 93  -  Version 1.0 completed.

         Notes:
                COMMENT CONVENTIONS:
                This comment format:
                ======================================================
                Category
                ======================================================
                indicates the start of a major category of markup
                declarations.

                This comment format:
                Group ================================================
                indicates a grouping of declarations within a major
                category.

                Right justified comments following an element
                declaration indicate the full name of the element,
                without abbreviation.  If the element name is not
                abbreviated, this comment will not appear.

                Right justified comments after an attribute
                declaration in a attribute list declaration give a very
                brief description of the content or function of the
                attribute.

                The comments are not intended to be a complete
                description of SGML BookMaster.  Refer to the
                SGML BookMaster Language Reference for complete
                descriptions of elements, their attributes and
                notations.

                CODING CONVENTIONS:
```

© Copyright IBM Corp. 1992, 1993

```
                Parameter entities are declared at the begining of the
                the DTD, unless they are intended to be used only once,
                in which case they are declared where they are used.

                Elements are declared as soon as possible after they are
                used, in the order they are used.  If multiple points
                are appropriate, a cross-reference will be used to
                direct the reader to the declaration of the element.

                Elements and their attributes are declared as closely as
                possible to each other.

                Attribute declarations are separated with a blank line
                into two types: user accessable and those which are
                defining to the system (architectural form attributes).

                A trick has been employed to force content in some
                contexts:  A required element with omissable start and
                end tags is defined to contain the required data.
                The tags for this element are not normally specified.
                If no data is provided and the tags are not specified,
                A validating parser will issue an error message since
                a start tag may not be implied for an empty element.

                The DTD's declarations conform to a specified format
                convention.  Feedback on the readability of the format
                used is welcome.
    ================================================================-->

<!--================================================================
    Parameter Entity Declarations
    ================================================================-->

<!--The following list defines those parameter entities which are
    defined so that the structure and content of this DTD may be
    modified, without modifying this DTD file itself.
    This facility is supported primarily for the inclusion of
    "towers" of functions, specification of more restrictive
    content models and and the addition or modification of the
    list of supported notations.  Here is the list of parameter
    entities defined for this purpose:

      To change the element groups allowed for particular elements:
        InfoUnitPool.Base - Elements unique to the InfoUnitPool base
        InfoUnitPool.Towers - InfoUnitPool elements used from the
             SGML BookMaster towers
        InfoUnitPool.DataPool.Common - Elements shared with DataPool
             This includes elements from the base and from the towers.
        DataPool.Base - Elements unique to the DataPool base
        DataPool.Towers - DataPool elements used from the SGML BookMaster
             towers

      To change the document structure elements' content:
        DocumentType - Name of the document type
        DocumentElement - Content model of the document element
        DocumentAtts - Attribute list of the document element
        Prolog - Content model of Prolog
        DocTrackingInfo - Content model of DocTrackingInfo
```

```
      FrontMatter - Content model of FrontMatter
      Body - Content model of Body
      Appendix - Content model of Appendix
      BackMatter - Content model of BackMatter
      Division - Content model of all hierarchical divisions
      SpecialDivision - Content model of all special divisions
      DivProlog - Content model of DivProlog
      SpecialDivProlog - Content model of SpecialDivProlog
      TrackingInfo - Content model of TrackingInfo

    To add or remove IBM-specific elements:
      BibRecord
      BOFNumber
      LibRecord
      PartNumber
      ProductInfo
      Safety
      Table

    To change the common attributes:
      Hyperlink.Attributes
      Property.Attributes
      Common.Atts.Base
      Common.Atts.NoID
      Common.Atts.NoConRef
      Common.Atts.NoHyNames
      Common.Atts.NoLexModel
      Common.Atts.NoRefType
      Common.Atts

    To change the notations supported by class:
      Math.Notations - Math and equation notations supported
      Graphics.Notations - Graphics and image notations supported
      ComputerLanguage.Notations - Computer language notations
           supported
      TextLanguage.Notations - Text notations supported

    To change the notations supported by various elements:
      Screen.Notations - Notations supported in Screen element
      LitData.Notations - Notations supported in the LitData element
      ObjectContainer.Notations - Notations supported in ObjContainer

    To change the table markup used (more than one may be used):
      BookMasterTable - IGNORE or INCLUDE
      SimpleTable - IGNORE or INCLUDE
      SoftQuadTable - IGNORE or INCLUDE

    To change the character entities used:
      UseBMSymbols - IGNORE or INCLUDE
      UseISOCharEnts - IGNORE or INCLUDE

    To change the character entity definitions used:
      BookMasterSymbols - define as an external parameter entity
      ISOCharEntities - define as an external parameter entity

-->

<!--Element Groups Parameter Entities ================================-->
```

```
                                <!---------- IBM-Specific Entities-->

<!ENTITY % BibRecord        "BibRecord | IBMBibRecord" >
<!ENTITY   BibRecord        "%BibRecord;" >

<!ENTITY % BOFNumber        "BOFNumber | IBMBOFNumber" >

<!ENTITY % LibRecord        "LibRecord | IBMLibRecord" >
<!ENTITY   LibRecord        "%LibRecord;" >

<!ENTITY % PartNumber       "PartNumber | IBMPartNumber" >

<!ENTITY % ProductInfo      "ProductInfo  | IBMProductInfo" >

<!ENTITY % Safety           "Safety | IBMSafety" >

<!-- This parameter entity must agree with marked section keywords which
     include or exclude the different table models.
     Notice that BookMasterTable and SimpleTable may not be used
     together.
-->
<!ENTITY % Table            "BookMasterTable | SoftQuadTable" >

                                <!-------------------- InfoUnitPool-->

<!--InfoUnitPool is the paragraph-level elements, essentially, division
    content.  Built from more detailed parameter entities.
-->

<!-- This entity contains those elements which are common between
     InfoUnitPool and DataPool.  The elements on the first line of the
     definition are from the base, the following lines each contain
     elements from a separate tower.
-->
<!ENTITY % InfoUnitPool.DataPool.Common
    "Link | nameloc | XRef |
     MultiMediaObject |
     FN | Citation |
     %BibRecord; |
     IRef | I1 | I2 | I3 |
     Formula |
     RefKey"
>

<!-- The following entity contains all the InfoUnitPool elements that
     are defined as towers.  Each line contains a separate tower.
-->
<!ENTITY % InfoUnitPool.Towers
    "Annot |
     BibList | FigureList | TableList | FootnoteList |
     CGraphic |
     GL |
     %LibRecord; |
     LERS |
     Lines |
     LitData |
     LongQuote |
```

```
     ManPage |
     MarkedNote | MarkList |
     MessageList |
     NoteList |
     ParmList |
     Procedure |
     RetrievalAlternatives |
     Screen | XMP |
     Syntax"
>

<!-- The following entity contains the base InfoUnitPool elements
     except P, Note and ParaBlock since these are included below.
     As currently setup, P, Note and ParaBlock are all part of the base.
-->
<!ENTITY % InfoUnitPool.Base
    "Bridge |
     DL | OL | UL |
     Figure | InfoModule |
     ModularInformation |
     Table"
>

<!-- The following 3 entities are used in content models of the DTD. -->
<!ENTITY % InfoUnitPool.NoPara
    "Note | Caution | Danger | Warning |
     (%InfoUnitPool.Base;) |
     %InfoUnitPool.DataPool.Common |
     %InfoUnitPool.Towers;"
>

<!ENTITY % InfoUnitPool.NoNote
    "P | ParaBlock |
     (%InfoUnitPool.Base;) |
     %InfoUnitPool.DataPool.Common |
     %InfoUnitPool.Towers;"
>

<!ENTITY % InfoUnitPool
    "P | ParaBlock | Note | Caution | Danger | Warning |
     (%InfoUnitPool.Base;) |
     %InfoUnitPool.DataPool.Common |
     %InfoUnitPool.Towers;"
>

                                  <!----------------------- DataPool-->

<!--DataPool defines the phrase-level elements.  Elements that are
    either DataPool or InfoUnitPool are defined separately.
-->

<!--DataPool defines the phrase-level elements.  Elements that are
    either DataPool or InfoUnitPool are defined separately.
-->

<!ENTITY % DataPool.Towers
    "Address |
     APL |
```

```
     Quote |
     Bin | Char | Decimal | Hex | Number | Octal |
     Date |
     GLEntry |
     MD |
     MV |
     PK | PV |
     SyntaxPhrase |
     ProcRef |
     XPH"
>

<!ENTITY % DataPool.Base    "#PCDATA | Phrase | DefinedTerm">

<!ENTITY % DataPool
    "%DataPool.Base; |
     %DataPool.Towers; |
     %InfoUnitPool.DataPool.Common;"
>

                                  <!--======================== InfoPool-->

<!--InfoPool is the combination of InfoUnitPool and DataPool
    InfoUnitPool.Base does not contain the elements shared between
    InfoUnitPool and DataPool.
-->

<!ENTITY % InfoPool
    "((%InfoUnitPool.Base;) | %InfoUnitPool.Towers | P | ParaBlock |
      Note |
      %DataPool;)*"
>

<!ENTITY % InfoPool.NoPara
    "((%InfoUnitPool.Base;) | %InfoUnitPool.Towers | Note |
      %DataPool;)*"
>

<!ENTITY % InfoPool.NoNote
    "((%InfoUnitPool.Base;) | %InfoUnitPool.Towers | P | ParaBlock |
      %DataPool;)*"
>

<!--Common Attribute Entities ========================================-->

<!ENTITY % Hyperlink.Attributes.BookMaster
   "Qualification    NAMES        #IMPLIED"
>

<!ENTITY % Hyperlink.AnchRoles.BookMaster
   "RevisionSpecification"
>

<!ENTITY % Hyperlink.AnchorRoles
   "Annotation | PropertySource | Alternative"
>

<!ENTITY % Hyperlink.Attributes
```

```
   "ContentAssociations
                    NAMES        #IMPLIED
    AnchorRoles     NAMES        #IMPLIED
    LinkEnds        NAMES        #IMPLIED
    %Hyperlink.Attributes.BookMaster;"
>

<!ENTITY % IBMProperty.Attributes
   "IBMSecurity     (UNC | IUO | IC | ICR | IRC)
                                 #IMPLIED"
>

<!ENTITY % Property.Attributes
   "Language        NAME         #IMPLIED
    Security        NAME         #IMPLIED
    Status          (New | Changed | Deleted | Unchanged)
                                 Unchanged
    Style           CDATA        #IMPLIED
    Version         CDATA        #IMPLIED
    Level           CDATA        #IMPLIED
    HWPlatform      CDATA        #IMPLIED
    SWPlatform      CDATA        #IMPLIED
    OutputDevice    CDATA        #IMPLIED
    %IBMProperty.Attributes;"
>

<!-- Definition of VersionSpec model

<HyLex ltn=VersionSpec>
     S,
     (("#NOT", S)?, name, S) |
     (VersionSpec, ("#AND" | "#OR"), VersionSpec) |
     ("(", VersionSpec, ")", S)
</HyLex>

-->

<!--Common.Atts.Base has all common attributes except those which
    are specified by any element in the DTD.  These attributes are:
        HyNames
        LexModel
        RefType
        ContentRef
        ID
-->

<!--The following entities are used where these attributes are overriden
    by specific elements to include the common specifications for the
    attribute.
-->

<!ENTITY   Common.HyNames
    "linkends LinkEnds anchrole AnchorRoles conloc ContentRef"
>

<!ENTITY   Common.LexModel
    "Version         VersionSpec
     Level           VersionSpec
```

```
     HWPlatform      VersionSpec
     SWPlatform      VersionSpec
     OutputDevice    VersionSpec
     docroles names"
>

<!ENTITY   Common.RefType
    "Qualification Qualification"
>

<!ENTITY % Common.Atts.Base
   "XRefText         CDATA        #IMPLIED
    InfoType         CDATA        #IMPLIED
    %Hyperlink.Attributes;
    %Property.Attributes;"
>

<!--Each of the following entities consists of Base, all of the
    excluded attributes but the one to be excluded, indicated in the
    name of the attribute.
-->

<!--The Class attribute defined here is used for local additions to
    this DTD.  It is our intent to perform this function using
    SUBDOC to isolate these additions and using architectural
    forms to define the processing of the element when this is
    technically possible (these functions are supported by vendors).
    concepts).
-->

<!ENTITY % Common.Atts.NoConRef
   "ID               NAME         #IMPLIED
    HyNames          CDATA        #FIXED    ''  -- no conref att here --
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoID
   "ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoLexMdl
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
```

```
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoLexMdl.NoHyTime
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoRefType
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoRefType.NoClass
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoClass
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    %Common.Atts.Base;"
>
<!ENTITY % Common.Atts.NoLexMdl.NoRefType
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts.NoHyTime
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
```

```
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!ENTITY % Common.Atts
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    HyTime           (ilink | hybrid)
                                  hybrid
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    %Common.Atts.Base;"
>

<!-- These attributes are common to BibRecord and LibRecord and
     their derivitives.  They are broken out to add the DescribedDocument
     AnchRole.  They are placed here to facilitate their use in all of
     these elements.
-->
<!ENTITY % LibRecord.Atts
   "ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED    '&Common.RefType;'
    Class            NAME         #IMPLIED
    XRefText         CDATA        #IMPLIED
    InfoType         CDATA        #IMPLIED
    ContentAssociations
                     NAMES        #IMPLIED
    AnchorRoles      NAMES        #IMPLIED
    LinkEnds         NAMES        #IMPLIED
    %Hyperlink.Attributes.BookMaster;
    %Property.Attributes;"
>


<!--Notation Entities ================================================-->

<!--Graphic.Notations defines the names of the graphic notations
    supported.
-->
<!ENTITY % Graphic.Notations
    "Image | Vector | PSEG | EPS | GDF | CGMChar | CGMClear | BMP | MET |
     TIFF | CDR"
>

<!--Math.Notations defines the names of the math and equation
    notations supported.
-->
<!ENTITY % Math.Notations   "TeXMath | eqn | ISOMath | smff" >

<!--ComputerLanguage.Notations defines the names of the computer
```

```
    language notations supported.  APL is excluded from this list
    it has its own element markup which uses the APL notation.
-->
<!ENTITY % ComputerLanguage.Notations
    "ASM | ASSEMBLE | C | EXEC | PLI | REXX"
>

<!--TextLanguage.Notations defines the names of the text markup
    language notations supported.
-->
<!ENTITY % TextLanguage.Notations "Script | TeXText" >

<!--%Screen.Notations; is defined before the Screen element
    declaration
-->

<!--%LitData.Notations; is defined before the LitData element
    declaration.
-->

<!--%ObjectContainer.Notations; is defined before the ObjectContainer
    element declaration.
-->

<!--Marked Section Control Entities ==================================-->

<!-- These marked section keyword status area values must agree with
     the contents of the %Table parameter entity declaration.
     Notice that BookMasterTable and SimpleTable may not be used
-->
<!ENTITY % BookMasterTable  "INCLUDE" >
<!ENTITY % SimpleTable      "IGNORE " >
<!ENTITY % SoftQuadTable    "INCLUDE" >


<!--=====================================================================
    Notation and Data Attribute Declarations
    =====================================================================-->

<!--Many of the following notation declarations need to have the
    appropriate version information added by an expert in these
    formats
-->

<!-- Within this DTD, notation attributes use a token list or the
     attribute declared value of NAME rather than NOTATION.  This is
     because of a conflict between our desire to specify default
     notations for some elements as well as provide a content reference
     capability for every element and the requirement in SGML (See
     7.9.4.4) that no notation be specified when a content reference
     attribute is used on the element.  Attributes which fall into this
     catagory are all named 'Notation'.  Should a conflict arise during
     processing between the notation specified for a referencing
     element and the reference target (be it an element or an entity),
     the target's notation will be used by application convention.
-->

<!-- Mathematics Notations ============================================-->
```

```
<!-- The following mathematics notation is IBM-defined notations. -->
<!NOTATION SMFF          PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Script Mathematical Formula Formatter//EN"
>

<!--The following mathematics notations are not defined by IBM, these
    identifiers will need to change when the registration of
    identifiers is implemented.
-->
<!NOTATION TeXMath       PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION TeX Mathematics Markup//EN"
>
<!NOTATION eqn           PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION eqn Mathematics Markup//EN"
>
<!--This mathematics notation is fundamentally different, it is
    defined by an SGML document type definition fragment.  This
    system's handling of this is not fully understood yet.
-->
<!NOTATION ISOMATH       PUBLIC
    "ISO/IEC TR9573-7//NOTATION Mathematics Markup//EN" >

<!-- Graphics and Image Notations ====================================-->

<!-- The following graphics notations are IBM-defined notations. -->
<!NOTATION PSEG          PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION AFP Picture Segment//EN"
>
<!NOTATION GDF           PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Graphic Data File//EN"
>
<!NOTATION MET           PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION OS/2 Metafile Format//EN"
>
<!NOTATION BMP           PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION OS/2 Bitmap Format//EN"
>

<!-- The following graphics notations are IBMIDDOC-defined.
     These notations indicate that the artwork is of a general class
     but that the specific notation may vary depending on the type
     of processing being defined and that the binding of the graphic to
     specific notation for processing should be delayed as long as
     possible. -->
<!NOTATION Graphics      PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Generic Graphics Meta-Notation//EN"
>
<!NOTATION Vector        PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Generic Vector Meta-Notation//EN"
>
<!NOTATION Image         PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Generic Image Meta-Notation//EN"
>

<!-- The following graphics notations are not defined by IBM, these
     identifiers will need to change when the registration of
```

```
     identifiers is implemented. -->
<!NOTATION CDR           PUBLIC
    "-//Corel//NOTATION CorelDraw File Format//EN"
>
<!NOTATION CGMChar       PUBLIC
    "ISO 8632-2//NOTATION CGM Character encoding//EN"
>
<!NOTATION CGMClear      PUBLIC
    "ISO 8632-4//NOTATION CGM Clear text encoding//EN"
>
<!NOTATION EPS           PUBLIC
    "-//Adobe//NOTATION Encapulated PostScript//EN"
>
<!NOTATION TIFF          PUBLIC
    "-//Aldus//NOTATION Tagged Image File Format//EN"
>

<!-- Computer Language Notations ====================================-->

<!NOTATION APL           PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION APL//EN"
>
<!NOTATION ASM           PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION 80x86 Assembler//EN"
>
<!NOTATION ASSEMBLE      PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION 370 Assembler//EN"
>
<!NOTATION C             PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION C//EN"
>
<!NOTATION EXEC          PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION EXEC//EN"
>
<!NOTATION PLI           PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION PL/I//EN"
>
<!NOTATION REXX          PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION REXX//EN"
>

<!-- Text Markup Language Notations ==================================-->

<!NOTATION SGMLDocument  PUBLIC
    "+//ISO/IEC 8879:1986//NOTATION SGML Document//EN"
>
<!ATTLIST  #NOTATION SGMLDocument
    HyTime           NAME         #FIXED    "document"
>
<!NOTATION HyQ           PUBLIC
    "+//ISO/IEC 10744colon.1992//NOTATION HyTime Query Notation//EN"
>
<!NOTATION HyLex         PUBLIC
    "+//ISO/IEC 10744colon.1992//NOTATION HyTime Lexical Model Notation//EN"
>

<!-- Within data of notation term, hyphen characters indicate
```

```
     allowed hyphenatation points.
-->
<!NOTATION Term          PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION DefinedTerm Content/Hyphenation Information//EN"
>
<!NOTATION TeXText       PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION TeX Text Markup//EN"
>
<!NOTATION Script        PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Document Composition Facility Markup//EN"
>

<!-- IBMIDDOC Special Text Notations ================================-->

<!-- Within data of notation LineSpecific, significant record ends
     must be preserved and presented, rather than being treated simply
     as word breaks.  It should be noted that without exception,
     significant record ends (line breaks) are treated as word breaks
     in IBMIDDOC.
-->
<!NOTATION LiteralData   PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Literal Data//EN"
>
<!NOTATION LineSpecific  PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Line Specific Content//EN"
>
<!NOTATION CGraphic      PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Character Graphic Drawing//EN"
>
<!NOTATION DTL           PUBLIC
    "+//ISBN 0-933186::IBM//NOTATION Dialog Tag Language//EN"
>

<!--GraphicNotation.Atts defines notation attributes for use with
     all graphic notations.
-->

<!ATTLIST  #NOTATION (%Graphic.Notations;)
    WIDTH            CDATA        #IMPLIED -- Width of graphic --
    DWidth           CDATA        #IMPLIED -- Width for online display --
    DEPTH            CDATA        #IMPLIED -- Depth of graphic --
    DDepth           CDATA        #IMPLIED -- Depth for online display --
    ALIGN            CDATA        #IMPLIED -- Horizontal alignment --
    BOTMAR           CDATA        #IMPLIED -- Margin below graphic --
    TOPMAR           CDATA        #IMPLIED -- Margin above graphic --
    ADJUST           CDATA        #IMPLIED -- Adjust rel to baseline --
    LABEL            CDATA        #IMPLIED -- Label for graphic --
>


<!--================================================================
    Document Type Organizational Elements
    ================================================================-->


<!ENTITY % DocumentType     "IBMIDDoc" >

<!ENTITY % DocumentElement
```

```
    "Prolog?, Frontmatter?, Body, WebSet?, Backmatter?"
>

<!ENTITY % Document.Atts
   '%Common.Atts.NoHyTime;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    HyTime           NAME         #FIXED    "HyDoc"     -- HyTime doc --
    unmspace         (unified | separate)
                                  #FIXED    "unified"'
>

<!ELEMENT  %DocumentType;   O O (%DocumentElement;)
                                             -- IBM Software Document --
>
<!ATTLIST  %DocumentType;
    %Document.Atts;
>

<!ENTITY % Prolog
    "DocTrackingInfo?, PropertyDefinitions?,  LinkDescriptions?,
     CopyRights?, TrademarkDefinitions?,  GlossaryDefinitions?,
     BibRecords?, IndexDefinitions?, ObjectLibrary*"
>

<!ELEMENT  Prolog           - O (%Prolog;) >
<!ATTLIST  Prolog
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Metadata"
>

<!-- '|' should be '&' and only Div allowed multiple times
     but tools don't properly support this *yet* -->
<!ENTITY % Frontmatter
    "(Abbreviations? | Abstract? | Bibliography? |
      EditionNotices? |
      FigureList? |
      Glossary? | Legend? | Preface? | SOA? | TableList? | TOC? |
      (%Safety;)? |
      Div
     )*"
>

<!ELEMENT  Frontmatter      - O (%Frontmatter;) >
<!ATTLIST  Frontmatter
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ENTITY % Body             "Title?, (Part* | (Div | Procedure)*)" >

<!ELEMENT  Body             O O (%Body;) >
<!ATTLIST  Body
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  Part             - O (Title?, Desc?, Div*) >
<!ATTLIST  Part
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ENTITY % Appendix         "Title?, Div*" >

<!ELEMENT  Appendix         - O (%Appendix;) >
<!ATTLIST  Appendix
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!-- '|' should be '&' and only Div allowed multiple times
     but tools don't properly support this *yet* -->
<!ENTITY % Backmatter
    "(Appendix? | Bibliography? | Glossary? | Index? | SOA? | Div)*"
>

<!ELEMENT  Backmatter       - O (%Backmatter;) >
<!ATTLIST  Backmatter
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!--Prolog Contents ================================================-->

<!--In the Prolog section, because of the large number of elements and
    the complexity of the markup, more commentary designed to help
    navigate is required.  Within this section, the following comment
    format indicates a prolog content major category:

    ============================================== PrologMajorCategory

    and the following comment format is used for prolog content
    sub categories:

                                     ============== PrologSubCategory
-->

<!--================================================= DocTrackingInfo-->

<!ENTITY % DocTrackingInfo
    "(%BibRecord;)?,
     ControlInfo?, RevisionHistory?, QualificationList?"
>

<!ELEMENT  DocTrackingInfo  - O
                     (%DocTrackingInfo;)
```

```
                                  -- Document Tracking Information --
>

<!ATTLIST  DocTrackingInfo
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Metadata"
>

                                   <!--======================== DocTitle-->
<!ELEMENT  DocTitle         - -
                     (ShortTitle?, Library?, Topic, SubTitle?)
                                              -- Document Title --
>
<!ATTLIST  DocTitle
    %Common.Atts;
    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "Title"
    Notation         (LineSpecific)
                                  #FIXED    "LineSpecific"
>

<!-- Note:  ShortTitle is the same element contained by Title -->

<!ELEMENT  (Library | Topic | SubTitle)
                            O O (%DataPool;)*
>
<!ATTLIST  (Library | Topic | SubTitle)
    %Common.Atts;
>

<!ELEMENT  VolumeIdentifier - O (%DataPool;)* >
<!ATTLIST  VolumeIdentifier
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

                                   <!--======================== Authors-->
<!ELEMENT  Authors          - - (Author)+ >
<!ATTLIST  Authors
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "Authors"
>

<!ELEMENT  Author           O O (Enterprise | Person) >
<!ATTLIST  Author
    Role             CDATA        #IMPLIED           -- Author's role --
    %Common.Atts;
>

<!ELEMENT  Enterprise       - O (EnterpriseName, Address?) >
<!ATTLIST  Enterprise
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "NameAndAddress"
>

<!ELEMENT  EnterpriseName   O O (%DataPool;)* >
<!ATTLIST  EnterpriseName
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Name"
>

<!ELEMENT  Person           - O (Name, Address?) >
<!ATTLIST  Person
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "NameAndAddress"
>

<!ELEMENT  Name             O O
                    (NameBlock? |
                     (Last, First?, Middle?, Honorific?,
                      JobTitle?, Nickname?, GoesBy?, Desc?))
>
<!ATTLIST  Name
    SortKey          CDATA        #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Name"
>

<!ELEMENT  NameBlock        - O
                    (%DataPool; | Last | First | Middle | Honorific |
                     JobTitle | Nickname | GoesBy | Desc)*
>
<!ATTLIST  NameBlock
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "NameBlock"
    Notation         (LineSpecific)
                                  LineSpecific
                        -- LineSpecific unless Object attribute used --
>

<!ELEMENT  (Last | First | Middle | Honorific |
            JobTitle | Nickname | GoesBy)
                          - O (%DataPool;)*
>
<!ATTLIST  (Last | First | Middle | Honorific |
            JobTitle | Nickname | GoesBy)
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "Name"
>

<!ELEMENT  Address          - -
                    (AddressBlock? |
```

```
                     (EnterpriseName?,
                      CareOf?, Street?, POBox?,
                      City?, State?, USZip?,
                      Country?,
                      Dept?, Bldg?, MailStop?,
                      Phone*,
                      VNET*, Internet*, IBMMail?, OLService*
                     ))
>
<!ATTLIST  Address
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Address"
>

<!ELEMENT  AddressBlock     - -
                     (%DataPool; | EnterpriseName |
                      CareOf | Street | POBox | City | State | USZip |
                      Country | Dept | Bldg | MailStop | VNET |
                      Internet | IBMMail | OLService | Phone)*
                                    -- Non-standard address block --
                     -- Use address subelements to do content markup. --
>
<!ATTLIST  AddressBlock
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "AddressBlock"
    Notation         (LineSpecific)
                                  LineSpecific
                         -- LineSpecific unless Object attribute used --
>

<!ELEMENT  (CareOf | Street | POBox | City | State | USZip |
            Country | Dept | Bldg | MailStop | VNET | Internet |
            IBMMail | OLService)
                               - O (%DataPool;)*
>

<!ATTLIST  (CareOf | Street | POBox | City | State | Country |
            Dept | Bldg | MailStop | IBMMail)
    %Common.Atts;
>

<!ELEMENT  Phone            - O (%DataPool;) >

<!ATTLIST  Phone
    Location         (Work | AltWork | Mobile | PDA | Home | AltHome |
                      OtherLoc)   Work
    Equipment        (Voice | VoiceMail | Fax | VoiceFax | VoiceMailFax |
                      Pager | Modem | Telex | OtherEquip)
                                  VoiceMail
    Coverage         (Primary | Group | Secretary | AnsweringService |
                      BBS | OtherCoverage)
                                  Primary
    Scope            (LocalPrivateNetwork | PrivateNetwork |
                      IBMTieLine | InState | Domestic | International |
```

```
                     OtherScope)
                                Domestic
    Use              (BusinessHours | AfterHours | AllHours |
                      Emergency | NoAnswer | OtherUse)
                                AllHours
    OtherInformation CDATA      #IMPLIED
    %Common.Atts;
>

<!ATTLIST  USZip
    %Common.Atts.NoLexMdl;

    LexModel         CDATA      #FIXED
                          "#CONTENT ([0-9],[0-9],[0-9],[0-9],[0-9],
                                     ('-',[0-9],[0-9],[0-9],[0-9])?
                                    )
                           &Common.LexModel;"
                         -- HyLex restriction: 7- or 9-digit ZIP code --
>

<!ATTLIST  VNET
    %Common.Atts.NoLexMdl;

    LexModel         CDATA      #FIXED
                          '#CONTENT ((name, S, "at", S, name) |
                                                (name, "(",name,")"))
                           &Common.LexModel;'
                     -- HyLex restriction: user at node or node(user) --
>

<!ATTLIST  Internet
    %Common.Atts.NoLexMdl;

    LexModel         CDATA      #FIXED
                          '#CONTENT (name, "@", name, (".",name)*)
                           &Common.LexModel;'
                              -- HyLex restriction: name@name(.name)* --
>

<!ATTLIST  OLService
    OLServiceName    CDATA      #IMPLIED
    %Common.Atts;
>

                                  <!--== Misc Document Identification-->
<!ELEMENT  FileNumber       - O (%DataPool;)* >
<!ATTLIST  FileNumber
    %Common.Atts;

    InfoMaster       NAME       #FIXED    "ContainingDivision"
>

<!ELEMENT  RetrievalKey     - O (%DataPool;)* >
<!ATTLIST  RetrievalKey
    %Common.Atts;

    InfoMaster       NAME       #FIXED    "ContainingDivision"
>
```

```
                                  <!--==================== ControlInfo-->
<!ENTITY % ControlInfo
    "(%ProductInfo;)*,
     Approvers?, Owners?,
     CriticalDates?"
>

<!ELEMENT  ControlInfo      - - (%ControlInfo;)
                                    -- Document Control Information --
>
<!ATTLIST  ControlInfo
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  ProductInfo      - O (ProductName, Version?) >
<!ATTLIST  ProductInfo
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT (ProductName | Version)
                           - O (%DataPool;)*
>
<!ATTLIST (ProductName | Version)
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  IBMProductInfo   - O (ProductName,
                                 Version?, Release?, Modification?,
                                 IBMProgramNumber?)
>
<!ATTLIST  IBMProductInfo
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT (Release | Modification | IBMProgramNumber)
                           - O (%DataPool;)*
>
<!ATTLIST (Release | Modification | IBMProgramNumber)
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  Approvers        - O (Enterprise | Person)+ >
<!ATTLIST  Approvers
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  Owners           - O (Enterprise | Person)+ >
<!ATTLIST  Owners
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  CriticalDates    - O (CriticalDate)+ >
<!ATTLIST  CriticalDates
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  CriticalDate     O O (Date, Desc) >
<!ATTLIST  CriticalDate
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

                                  <!--================ RevisionHistory-->
<!ELEMENT  RevisionHistory  - - (Revision | Mark)+ >
<!ATTLIST  RevisionHistory
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  Revision         - O (Desc, Author, Date)
                                             -- Revision definition --
>
<!ATTLIST  Revision
    ID               NAME         #REQUIRED
    TrackingCode     CDATA        #IMPLIED  -- Revision tracking code --
    Reason           CDATA        #IMPLIED
                   -- Reason for revision: editorial, technical, etc. --
    %Common.Atts.NoID;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

                                     <!--============ QualificationInfo-->
<!ELEMENT  QualificationList
                             - - (Qualification)+ >
<!ATTLIST  QualificationList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  Qualification    - - (Desc)    -- Qualification definition -->
<!ATTLIST  Qualification
    ID               NAME         #REQUIRED
    Run              (Run | Ignore)
```

```
                                Run
    QualificationText
                    CDATA       #IMPLIED
                                  -- Qualification identifying text --
    %Common.Atts.NoID;

    InfoMaster      NAME        #FIXED    "ContainingDivision"
>

<!--============================================== PropertyDefinitions-->

<!ENTITY % PropertyDefinitions
    "PropertyGroup |
     PropertyDef | AnchorRoleDef | ClassDef | LERSDef | ManPageDef |
     MarkDesc | MessageItemDef | ModularInfoDef | ModularItemDef"
>

<!ELEMENT  PropertyDefinitions
                             - - (%PropertyDefinitions;)*
>
<!ATTLIST  PropertyDefinitions
    %Common.Atts;

    InfoMaster      NAME        #FIXED    "ContainingDivision"
>

<!ELEMENT  PropertyGroup    - - (Desc?, (PropertyDef* | PropertyGroup*))>
<!ATTLIST  PropertyGroup
    %Common.Atts;
>

<!ELEMENT  PropertyDef      - O (desc?)         -- Property Defintion -->
<!ATTLIST  PropertyDef
    ElementTypes    NAMES       #IMPLIED
    %Common.Atts;

    InfoMaster      NAME        #FIXED    "PropertyDefinition"
>

<!ELEMENT  AnchorRoleDef    - - (Title?, Semantic)
                                            -- Class Definition --
>
<!ATTLIST  AnchorRoleDef
    RoleName        NAME        #REQUIRED
    ParentRole      NAME        #IMPLIED    -- Parent of this class --
    ElementTypes    NAMES       #IMPLIED -- GIs this Def applies to --
    %Common.Atts;

    InfoMasterSemanticClass
                    NAME        #FIXED    "Content"
    InfoMaster      NAME        #FIXED    "PropertyDefiniton"
>

<!ENTITY % ClassDef         "Title?, IndexTerm?, Semantic" >

<!ELEMENT  ClassDef         - - (%ClassDef;)      -- Class Definition -->
<!ATTLIST  ClassDef
    ClassName       NAME        #REQUIRED
```

```
    ParentClass      NAME         #IMPLIED    -- Parent of this class --
    ElementTypes     NAMES        #IMPLIED -- GIs this Def applies to --
    SemanticClass    NAME        -- (Content | MetaData | Annotation) --
                                  MetaData
                                  -- The semantic class of this class --
    SemanticProperty (PropertyDefinition | Enumeration)
                                  #IMPLIED
    %Common.Atts;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Content"
    InfoMaster       NAME         #FIXED    "PropertyDefiniton"
>
<!ELEMENT  Semantic         - O (%DataPool;)* >
<!ATTLIST  Semantic
    %Common.Atts;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!ENTITY % lersdef.atts
   "Auth             CDATA        #IMPLIED
    Comments         CDATA        #IMPLIED
    Context          CDATA        #IMPLIED
    Defaults         CDATA        #IMPLIED
    Errcond          CDATA        #IMPLIED
    Examples         CDATA        #IMPLIED
    Flags            CDATA        #IMPLIED
    Format           CDATA        #IMPLIED
    Intrep           CDATA        #IMPLIED
    Messages         CDATA        #IMPLIED
    Other            CDATA        #IMPLIED
    Parms            CDATA        #IMPLIED
    Process          CDATA        #IMPLIED
    Purpose          CDATA        #IMPLIED
    Restrict         CDATA        #IMPLIED
    Results          CDATA        #IMPLIED
    Retcodes         CDATA        #IMPLIED
    Sysenv           CDATA        #IMPLIED
    Usage            CDATA        #IMPLIED
    Ver              CDATA        #IMPLIED"
>
<!ELEMENT  LERSDef          - O (Desc?)
                     -- Language Element Reference Section Definition --
>
<!ATTLIST  LERSDef
    %lersdef.atts;
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "PropertyDefinition"
>

<!ELEMENT  ManPageDef       - O (Desc?)        -- Man Page Definition -->
<!ATTLIST  ManPageDef
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "PropertyDefinition"
>

<!ELEMENT  ModularInfoDef   - O (Desc, ModularItemDef+)
                                  -- Module Information Definition --
>
<!ATTLIST  ModularInfoDef
    ClassName        NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "PropertyDefinition"
>

<!-- Title element defines default title for a given ModuleItem class -->
<!ELEMENT  ModularItemDef   - O (Title, Desc?)
                                       -- Module Item Definition --
>
<!ATTLIST  ModularItemDef
    ClassName        NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "PropertyDefinition"
>

<!--================================================ LinkDescriptions-->

<!ELEMENT  LinkDescriptions - O (LinkDesc*) >
<!ATTLIST  LinkDescriptions
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  LinkDesc         - - (Desc?, (nameloc | nmquery)?, NotLoc*)
                                        -- Link Description --
>
<!ATTLIST  LinkDesc
    %Common.Atts.NoHyTime;

    InfoMaster       NAME         #FIXED    "LinkDescription"

    -- HyTime attributes: --
    HyTime           NAME         #FIXED    nameloc
    ordering         (ordered)    ordered
              -- Up to presentation system to define meaning of order --
    set              (set)        set                -- Ignore dupes --
    aggloc           (nagg)       nagg              -- Not aggregate --
    spanloc          (nspan)      nspan                  -- No spans --
    limsort          (nlimsort)   nlimsort       -- No limit sorting --
>

<!ELEMENT  nameloc          - O (nmquery | nmlist)*
                                       -- HyTime nameloc element --
>
<!ATTLIST  nameloc
    ID               NAME         #REQUIRED

             -- HyTime attributes: --
```

```
    HyTime          NAME        #FIXED    namloc
    ordering        (ordered)   ordered
              -- Up to presentation system to define meaning of order --
    set             (set)       set                -- Ignore dupes --
    aggloc          (nagg)      nagg              -- Not aggregate --
    spanloc         (nspan)     nspan                  -- No spans --
    limsort         (nlimsort)  nlimsort       -- No limit sorting --
>

<!ELEMENT  nmlist          - O (#PCDATA)        -- HyTime name list -->
<!ATTLIST  nmlist
    -- HyTime attributes: --
    HyTime          NAME        #FIXED    nmlist
    nametype        (entity | element)
                                element     --Unified not supported --
    obnames         (nobnames)  nobnames
    docname         ENTITY      #IMPLIED
    dtdorlpd        NAMES       #IMPLIED
    HyNames         CDATA       #FIXED    "docorsub docname"
>

<!ELEMENT  nmquery         - O (#PCDATA)       -- HyTime name query -->
<!ATTLIST  nmquery
    fn       -- Function name --
             -- Constraint: unique in element type --
                    NAME     #IMPLIED  -- Default: none --
    usefn    -- Fn of another function to which to pass
                the args of this one --
                    NAME     #CONREF   -- Default: none --
    args     -- Argument string passed to the function
                (if supported by function language) --
                    CDATA    #IMPLIED  -- Default: none --

    ID              NAME        #IMPLIED
    notation        NAME        #IMPLIED  -- Default is HyQ --
    -- HyTime attributes: --
    HyTime          NAME        #FIXED    nmquery
    qdomain         NAMES       #IMPLIED
    qcontext        NAME        #IMPLIED
>

<!ELEMENT  NotLoc          - - (#PCDATA) -- HyTime notation location -->
<!ATTLIST  NotLoc
    ID              NAME        #REQUIRED
    Notation        NAME        #REQUIRED
    LocSrc          NAME        #REQUIRED

    -- HyTime attributes: --
    HyTime          NAME        #FIXED    "notloc"
    qdomain         NAMES       #IMPLIED
    qcontext        NAME        #IMPLIED
    ordering        (ordered)   ordered
              -- Up to presentation system to define meaning of order --
    set             (set)       set                -- Ignore dupes --
    aggloc          (nagg)      nagg              -- Not aggregate --
>

<!--================================================== CopyRights-->
```

```
<!ELEMENT  CopyRights       - - (CopyRight+) >
<!ATTLIST  CopyRights
    %Common.Atts;
    InfoMethod       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  CopyRight        - -
                    (Holder, FirstYear, LastYear?, RightsReserved?,
                     RightsGiven?, USGovtLicense?,
                     RegisteredWith?, CopyRightExt?)
>
<!ATTLIST  CopyRight
    %Common.Atts;
>

<!ELEMENT  Holder           - O (%DataPool;)* >
<!ATTLIST  Holder
    %Common.Atts;
>

<!ELEMENT  (FirstYear | LastYear)
                            - O (%DataPool;)*
>
<!ATTLIST  (FirstYear | LastYear)
    %Common.Atts;
>

<!ELEMENT  RightsReserved   - O (%DataPool;)* >
<!ATTLIST  RightsReserved
    %Common.Atts;
>

<!ELEMENT  RightsGiven      - O (%DataPool;)* >
<!ATTLIST  RightsGiven
    %Common.Atts;
>

<!ELEMENT  USGovtLicense    - O (%DataPool;)* >
<!ATTLIST  USGovtLicense
    Auto             (Auto)       #CONREF
    %Common.Atts;
>

<!ELEMENT  RegisteredWith   - O (%DataPool;)* >
<!ATTLIST  RegisteredWith
    %Common.Atts;
>

<!ELEMENT  CopyrightExt     - O (%InfoUnitPool;)*
                                              -- Copyright extension --
>
<!ATTLIST  CopyrightExt
    %Common.Atts;
>

<!--============================================= TradeMarkDefinitions-->
```

```
<!ELEMENT  TradeMarkDefinitions
                            - O (TradeMark*)
>
<!ATTLIST  TradeMarkDefinitions
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!-- Temporary definition of TradeMark to further DTD work. -->
<!ELEMENT  TradeMark        - O (#PCDATA) >

<!ATTLIST  TradeMark
    %Common.Atts;
>

<!--============================================== GlossaryDefinitions-->

<!ELEMENT  GlossaryDefinitions
                            - O (GLEntry*)
>
<!ATTLIST  GlossaryDefinitions
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!--======================================================= BibRecords-->

<!ELEMENT  BibRecords       - - (%BibRecord; | %LibRecord;)*
                                  -- Bibliographic record --
>
<!ATTLIST  BibRecords
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

                               <!--======= IBMBibRecord & BibRecord-->
<!ELEMENT  BibRecord        - O
                     (DocTitle,
                      Authors?,
                      Publisher?, PrintLocation?,
                      OrderNumber?,
                      ISBN?, PublicIdentifier?, Desc?)
                                  -- Bibliographic Record --
>
<!ATTLIST  BibRecord
    %LibRecord.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
    ApplicationClass NAME         #FIXED    "BibliographicRecord"
>

<!ELEMENT  IBMBibRecord     - O
                     (DocTitle, VolumeIdentifier?, RetrievalKey?,
                      Authors?,
                      Publisher?, PrintLocation?,
```

```
                    FileNumber?,
                    IBMDocNumber?, IBMPartNumber?, ISBN?,
                    PublicIdentifier?, Desc? )
                                    -- IBM Bibliographic Record --
>
<!ATTLIST  IBMBibRecord
    %LibRecord.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
    ApplicationClass NAME         #FIXED    "BibliographicRecord"
>

<!ELEMENT  Publisher        - O (EnterpriseName, Address?) >
<!ATTLIST  Publisher
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
             -- This element also conforms to the NameAndAddress form --
    ApplicationClass NAME         #FIXED    "Publisher"
>

<!ELEMENT  PrintLocation    - O (%DataPool;)* >
<!ATTLIST  PrintLocation
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "PrintLocation"
>

<!ELEMENT  OrderNumber      - O (#PCDATA)    -- Customer Order Number -->
<!ATTLIST  OrderNumber
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "OrderNumber"
>

<!ELEMENT  IBMDocNumber     - O (#PCDATA)      -- IBM Document Number -->
<!ATTLIST  IBMDocNumber
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "IBMDocNumber"
    LexModel         CDATA        #FIXED
       '(("SC" | "GC" | "LY" | "ZZ"), digit, digit, "-", digit,
         digit, digit, digit, "-", digit, digit)
        &Common.LexModel;'
>

<!ELEMENT  PartNumber       - O (#PCDATA) >
<!ATTLIST  PartNumber
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "PartNumber"
>
```

```
<!ELEMENT  IBMPartNumber    - O (#PCDATA) >
<!ATTLIST  IBMPartNumber
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "IBMPartNumber"
    LexModel         CDATA        #FIXED
       '(digit, digit, digit, digit, digit, digit, digit)
        &Common.LexModel;'
>

<!ELEMENT  ISBN             - O (#PCDATA) >
<!ATTLIST  ISBN
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ISBN"
    LexModel         CDATA        #FIXED
       '(digit, "-", digit, digit, "-", digit, digit, digit, digit,
         digit, digit, ("-", digit)?)
        &Common.LexModel;'
>

<!ELEMENT  PublicIdentifier - O (#PCDATA) -- Formal Public Identifier -->
<!ATTLIST  PublicIdentifier
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "PublicIdentifier"
    LexModel         CDATA        #FIXED
       (("-" | "+"), "//", owneridentifier, "//", publictextidentifier,
        "//", languagecode)
        &Common.LexModel;'
>

                                  <!-------- IBMLibRecord & LibRecord-->
<!ELEMENT  LibRecord        - -
                     (Title, Publisher?, PrintLocation?,
                      BOFNumber?, PartNumber?,
                      Product*, ISBN?, PublicIdentifier?,
                      ContainedDocuments?, Desc?)
                                                 -- Library Record --
>
<!ATTLIST  LibRecord
    %LibRecord.Atts;
>

<!ELEMENT  IBMLibRecord     - -
                     (Title, Publisher?, PrintLocation?,
                      IBMBOFNumber?, IBMPartNumber?,
                      Product*, ISBN?, PublicIdentifier?,
                      ContainedDocuments?, Desc?)
>
<!ATTLIST  IBMLibRecord
    %LibRecord.Atts;
>

<!ELEMENT  ContainedDocuments
```

```
                            - O (#PCDATA | Citation )*
>
<!ATTLIST  ContainedDocuments
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                         "#CONTENT (name | #GI) &Common.LexModel;"
    RefType          CDATA        #FIXED
                         "#CONTENT (&BibRecord; | &LibRecord;)
                          &Common.RefType;"
            -- The order BibRecord IDs are specified defines the
               default order for presenting the list of documents
               in a generated bibliography --
>

<!ELEMENT  BOFNumber        - O (%DataPool;)* -- Bill of Forms number -->
<!ATTLIST  BOFNumber
    %Common.Atts;
>

<!ELEMENT  IBMBOFNumber     - O (%DataPool;)*
                                        -- IBM Bill of Forms number --
>
<!ATTLIST  IBMBOFNumber
    %Common.Atts;
>

<!ELEMENT  Product          - O (%DataPool;)* >
<!ATTLIST  Product
    %Common.Atts;
>

<!--================================================== IndexDefinitions-->

<!ELEMENT  IndexDefinitions - O (I1*) >
<!ATTLIST  IndexDefinitions
    %Common.Atts.NoConRef;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!--================================================== ObjectLibrary-->

<!ELEMENT  ObjectLibrary    - O (Desc?, ObjectLibraryBody) >
<!ATTLIST  ObjectLibrary
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  ObjectLibraryBody
                           O O ANY
>
<!ATTLIST  ObjectLibraryBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Container"
>
```

```
<!ELEMENT  RetrievalAlternatives
                           - O (Desc?, RetrievalAltsBody) >
<!ATTLIST  RetrievalAlternatives
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  RetrievalAltsBody
                           O O (%InfoPool;)
>
<!ATTLIST  RetrievalAltsBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Container"
>

<!--General Division Content ============================================-->

<!ENTITY % Division         "DivProlog, DivIntro?, DivBody, DivSummary?">

<!ENTITY % SpecialDiv
    "SpecialDivProlog, DivIntro?, DivBody, DivSummary?"
>

<!ELEMENT  Div              - - (%Division;) -- Hierarchical Division -->
<!ATTLIST  Div
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ENTITY % DivProlog
    "Title, Desc?, TrackingInfo?, PropertyDefinitions?,
     LinkDescriptions?, CopyRights?, TrademarkDefinitions?,
     GlossaryDefinitions?, BibRecords?, IndexDefinitions?,
     ObjectLibrary*"
>

<!ELEMENT  DivProlog        O O (%DivProlog;)      -- Division Prolog -->
<!ATTLIST  DivProlog
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Metadata"
>

<!ENTITY % TrackingInfo
    "Authors?, ControlInfo?, RevisionHistory?, QualificationList?"
>

<!ELEMENT  TrackingInfo     - - (%TrackingInfo;)
                                  -- Division Tracking Information --
>
<!ATTLIST  TrackingInfo
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "ContainingDivision"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Metadata"
>

<!-- Permissive model required to allow Div and Procedure to mingle. -->
<!ELEMENT  DivBody          O O (%InfoUnitPool; | Div)*
                                                -- Division Body --
>
<!ATTLIST  DivBody
    %Common.Atts;
    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  DivIntro         - O (Title?, TOC?, (%InfoUnitPool;)*, Div*)
                                          -- Division Introduction --
>
<!ATTLIST  DivIntro
    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
    %Common.Atts;
>

<!ELEMENT  DivSummary       - O (Title?, (%InfoUnitPool;)*, Div*)
                                                -- Division Summary --
>
<!ATTLIST  DivSummary
    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
    %Common.Atts;
>

<!--Specialized Division Elements ====================================-->

<!ENTITY % SpecialDivProlog
    "(GeneratedTitle | Title), Desc?, TrackingInfo?,
     PropertyDefinitions?, LinkDescriptions?, CopyRights?,
     TrademarkDefinitions?, GlossaryDefinitions?, BibRecords?,
     IndexDefinitions?, ObjectLibrary*"
>

<!ELEMENT  SpecialDivProlog O O (%SpecialDivProlog;)
                                                 -- Division Prolog --
>
<!ATTLIST  SpecialDivProlog
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Metadata"
>

<!ELEMENT  GeneratedTitle   - O EMPTY >
<!ATTLIST  GeneratedTitle
    %Common.Atts.NoConRef;
>

<!ELEMENT  Abbreviations    - O (%SpecialDiv;) >
```

```
<!ATTLIST  Abbreviations
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ELEMENT  Abstract         - O (%SpecialDiv;) >
<!ATTLIST  Abstract
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ELEMENT  Bibliography     - O (%SpecialDiv;) >
<!ATTLIST  Bibliography
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ELEMENT  EditionNotices   - O (%SpecialDiv;) >
<!ATTLIST  EditionNotices
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ELEMENT  FigureList       - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool)*)
                                )
>
<!ATTLIST  FigureList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>

<!ELEMENT  Glossary         - O (%SpecialDiv;) >
<!ATTLIST  Glossary
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ELEMENT  FootnoteList     - O (Title?,
                                 (nmquery | FN)*)
>
<!ATTLIST  FootnoteList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>

<!ELEMENT  IBMSafety        - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool;)*)
                                )
>
<!ATTLIST  IBMSafety
```

```
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>

<!ELEMENT  Index            - O (%SpecialDiv;) >
<!ATTLIST  Index
    Auto             (Auto)       #CONREF  -- Generated index --
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ELEMENT  Legend           - O (%SpecialDiv;) >
<!ATTLIST  Legend
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ELEMENT  MarkList         - O (Title?,
                                 (nmquery | (%InfoUnitPool;)*)
                                )
>
<!ATTLIST  MarkList
    ID               NAME         #IMPLIED
    Classes          NAMES        #REQUIRED
    Actions          NAMES        #REQUIRED
    Display          (Class | Action | Location | Item | Desc)
                                  #IMPLIED
    ClassHD          CDATA        #IMPLIED --Default def'd in doc style--
    ActionHD         CDATA        #IMPLIED --Default def'd in doc style--
    ItemHD           CDATA        #IMPLIED --Default def'd in doc style--
    LocationHD       CDATA        #IMPLIED --Default def'd in doc style--
    DescHD           CDATA        #IMPLIED --Default def'd in doc style--
    MarkIDs          NAMES        #REQUIRED   -- IDs of MARK elements --
    %Common.Atts.Base;

    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED
                          "MarkIDs Mark &Common.RefType"
    InfoMaster       NAME         #FIXED    "QueryLink"
>

<!ELEMENT  Preface          - O (%SpecialDiv;) >
<!ATTLIST  Preface
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>

<!ELEMENT  SOA              - O (%SpecialDiv;) >
<!ATTLIST  SOA
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "HierarchicalDivision"
>
```

```
<!ELEMENT  Safety           - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool;)*)
                                )
>
<!ATTLIST  Safety
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>

<!ELEMENT  TableList        - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool;)*)
                                )
>
<!ATTLIST  TableList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>

<!ELEMENT  TOC              - O ((GeneratedTitle | Title),
                                 (nmquery | (%InfoUnitPool;)*)
                                )
                                              -- Table of Contents --
>
<!ATTLIST  TOC
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "QueryLink"
>

<!--Hypertext Web Elements ====================================-->

<!ELEMENT  WebSet           - O (Web+)     -- Contains hyperlink webs -->
<!ATTLIST  WebSet
    %Common.Atts;
>

<!ELEMENT  Web              - O (DivProlog, WebNode+)
                                           -- Contains web nodes --
>
<!ATTLIST  Web
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  WebNode          - O (%Division;)   -- Contains a web node -->
<!ATTLIST  WebNode
    ID               NAME         #REQUIRED
    %Common.Atts.NoID;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!--================================================================
    Information Unit Pool Elements
```

```
    ================================================================-->

<!--Paragraph ============================================================-->

<!ELEMENT  P               - O (Title?, ParaBody)       -- Paragraph -->
<!ATTLIST  p
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
>
<!ELEMENT  ParaBody        O O (%InfoPool.NoPara;)*
                                          -- paragraph body --
>
<!ATTLIST  ParaBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ParagraphBody"
>

<!ELEMENT  ParaBlock       - - (Title?, (%InfoUnitPool;))*
                                        -- Paragraph block --
>
<!ATTLIST  ParaBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  Note            - O (Title?, NoteBody) >
<!ATTLIST  Note
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
>

<!ELEMENT  NoteBody        O O (%InfoPool.NoNote;)* >
<!ATTLIST  NoteBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ParagraphBody"
>

<!ELEMENT  Caution         - O (Title?, NoteBody) >
<!ATTLIST  Caution
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!ELEMENT  Danger          - O (Title?, NoteBody) >
<!ATTLIST  Danger
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
```

```
                    NAME         #FIXED    "Annotation"
>

<!ELEMENT  Warning          - O (Title?, NoteBody) >
<!ATTLIST  Warning
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!--Lists ===========================================================-->

<!ELEMENT  OL               - - (Title?, (LI | LIBlock | Bridge)+)
                                                 -- Ordered List --
>
<!ATTLIST  OL
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "SinglePartList"
    InfoMasterSemanticProperty
                     NAME         #FIXED    "Enumeration"
>

<!ELEMENT  UL               - - (Title?, (LI | LIBlock | Bridge)+)
                                               -- Unordered List --
>
<!ATTLIST  UL
    %Common.Atts

    InfoMaster       NAME         #FIXED    "SinglePartList"
>

<!ELEMENT  NoteList         - - (Title?, (LI | LIBlock | Bridge)+) >
<!ATTLIST  NoteList
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "SinglePartList"
    InfoMasterSemanticProperty
                     NAME         #FIXED    "Enumeration"
>

<!ELEMENT  LIBlock          - O (Title?, (LI | Bridge)+)
>
<!ATTLIST  LIBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ListItemBlock"
>

<!ELEMENT  LI               - O (%InfoPool;)*            -- List Item -->
<!ATTLIST  LI
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ListItem"
>
```

```
<!ELEMENT  DL              - - (Title?, (DTHD, DDHD)?,
                                (DLEntry | DLBlock | Bridge)+)
                                                -- Definition List --
>
<!ATTLIST  DL
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefiningList"
>

<!ELEMENT DTHD             - O (%DataPool;)*
                                -- Definition List Term Heading --
>
<!ATTLIST DTHD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermHeading"
>

<!ELEMENT DDHD             - O (%DataPool;)*
                               -- Definition List Definition Heading --
>
<!ATTLIST DDHD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionHeading"
>

<!ELEMENT  DLBlock         - O (Title?, (DLEntry | Bridge)+)
                                 -- Definition List Block of Entries --
>
<!ATTLIST  DLBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionBlock"
>

<!ELEMENT  DLEntry         - O (DefinedTerm, DD)
                                              -- Definition List Entry --
>
<!ATTLIST  DLEntry
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Term"
>

<!ELEMENT  DD              - O (%InfoPool;)*
                                -- Definition list term definition --
>
<!ATTLIST  DD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermDefinition"
>

<!ELEMENT  ParmList        - O (Title?, (PTHD, PDHD)?,
                               (nmquery |
                                (Parm | ParmBlock | Bridge)+
```

```
                                ))
                                        -- Parameter Definition List --
>
<!ATTLIST  ParmList
    GenerateFrom     NAME         #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefiningList"
>

<!ELEMENT PTHD              - O (%DataPool;)* >
<!ATTLIST PTHD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermHeading"
>

<!ELEMENT PDHD              - O (%DataPool;)* >
<!ATTLIST PDHD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionHeading"
>

<!ELEMENT  Parm             - O (PT, PD)
                                -- Parmeter Definition List Entry --
>
<!ATTLIST  Parm
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Term"
>

<!ELEMENT  PT               O O (%InfoPool;)*  -- Parameter List Term -->
<!ATTLIST  PT
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinedTerm"
>

<!ELEMENT  PD               - O (%InfoPool;)*
                                -- Parameter List Term Definition --
>
<!ATTLIST  PD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermDefinition"
>

<!ELEMENT  ParmBlock        - O (Title?, (Parm | Bridge)+)
                               -- Parameter List Block of Entries --
>
<!ATTLIST  ParmBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionBlock"
>
```

```
<!ELEMENT  GL              - - (Title?,
                                (nmquery |
                                 (GLEntry | GLBlock | Bridge)*
                               ))
                                                  -- Glossary List --
>
<!ATTLIST  GL
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefiningList"
>

<!ELEMENT  GLBlock         - O (Title?, (GLEntry | Bridge)+)
                                  -- Glossary List Block of Entries --
>
<!ATTLIST  GLBlock
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinitionBlock"
>

<!ELEMENT  GLEntry         - O (DefinedTerm, GD+)
                                               -- Glossary List Entry --
>
<!ATTLIST  GLEntry
    SeeAlso          NAMES        #IMPLIED
    SortText         CDATA        #IMPLIED
    %Common.Atts.NoRefType;

    InfoMaster       NAME         #FIXED    "Term"
    RefType          CDATA        #FIXED
                          "SeeAlso GLEntry &Common.RefType;"
>
<!ELEMENT  GD              - O (%InfoPool;)*
                                    -- Glossary List Term Definition --
>
<!ATTLIST  GD
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TermDefinition"
>

<!--Figure ============================================================-->

<!ELEMENT  Figure          - -
                     (Title?, Caption?, Desc?,
                      (FigSeg* | (%InfoUnitPool; | %Table;)*)
                     )
>
<!ATTLIST  Figure
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Display"
>

<!ELEMENT  Table           - -
                     (Title?, Caption?, Desc?,
                      (FigSeg* | (%InfoUnitPool; | %Table;)*)
```

```
                    )
>
<!ATTLIST  Table
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Display"
>

<!ELEMENT  FigSeg           - - (%InfoUnitPool; | %Table;)* >
<!ATTLIST  FigSeg
    InfoMaster       NAME         #FIXED    "FigureSegment"
    %Common.Atts;
>

<!--Table ============================================================-->

<![ %SimpleTable; [
<!ELEMENT  SimpleTable      - - (THeading?, TFooting?, TBody+) >
<!ATTLIST  SimpleTable
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Table"
>

<!ELEMENT  TBody            O O     (THeading?, TFooting?, TRow+)
                                                -- Table Body --
>
<!ATTLIST  TBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TableBody"
>

<!ELEMENT  THeading         - O (Row+)              -- Table Heading -->
<!ATTLIST  THeading
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TableHeading"
>

<!ELEMENT  TFooting         - O (Row+)              -- Table Footing -->
<!ATTLIST  TFooting
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TableFooting"
>

<!ELEMENT  TRow             - O (Title?, Cell+) >
<!ATTLIST  TRow
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "TableRow"
>

<!ELEMENT  Cell             - O (%InfoPool;)* >
<!ATTLIST  Cell
    N                NMTOKEN      #IMPLIED               -- Cell name --
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "TableCell"
>
]]>

<![ %BookMasterTable; [
<!ELEMENT  BookMasterTable  - - (THD?, TFT?, (Row | TNote)+) >
<!ATTLIST  BookMasterTable
    %Common.Atts;
    -- All BookMaster presentation attributes are placed in Style att --
    -- Reference to TDef BookMaster elements now use PropertyDef mech --
    -- TCap and TDesc info is now input in the Figure element content --

    InfoMaster       NAME         #FIXED    "BookMasterTable"
>

<!ELEMENT (THD | TFT | Row)
                             - O (C+)
>
<!ATTLIST (THD | TFT | Row)
    PropertyScope    (Temp)       #IMPLIED
    %Common.Atts;
>

<!ELEMENT  TNote             - O (%InfoPool;)* >
<!ATTLIST  TNote
    %Common.Atts;
>

<!ELEMENT  C                 - O (%InfoPool;)* >
<!ATTLIST  C
    N                NUMBER       #IMPLIED
    %Common.Atts;
>
]]>

<![ %SoftQuadTable; [
<!--
    SoftQuad's Canonical Table DTD fragment
    Copyright (c) 1991 by SoftQuad, Inc.

    This material is used with permission and may be reproduced in whole
    without restriction provided that this copyright notice is included.

    This DTD fragment defines the table structure used internally by
    SoftQuad Author/Editor.

    The names of the table elements and the content model for TblCell
    can be changed by predefining the appropriate parameter entities
    before including this DTD fragment. For example:

        <!ENTITY % TblBody  "(TblHead | TblBody)">
        <!ENTITY % CellMdl  "(#PCDATA)">
        <!ENTITY % tblDTD   SYSTEM "table.dtd">
        %tblDTD;

    will define two elements, TblHead and TblBody, each with the same
    content model as the canonical TblBody element, and it will change
```

```
    the content model of table cells from the default of ANY to
    character data.
-->

<!ENTITY % TblBody          "(TBLHeading | TBLBody | TBLFooting)" >
<!ENTITY % TblCDfs          "TBLCDefs" >
<!ENTITY % TblCDef          "TBLCDef" >
<!ENTITY % TblRows          "TBLRows" >
<!ENTITY % TblRow           "TBLRow" >
<!ENTITY % TblCell          "TBLCell" >
<!ENTITY % CellMdl          "(%InfoPool;)*"
                                 -- Same cell content as other tables --
>

<!ENTITY % VBorder
    "(VSingle|VDouble|VTriple|VDash|VDot|VBold|VBlank|VNone)"
>
<!ENTITY % HBorder
    "(HSingle|HDouble|HTriple|HDash|HDot|HBold|HBlank|HNone)"
>
<!ENTITY % VAlign  "(Top|Middle|Bottom)" >
<!ENTITY % HAlign  "(Left|Both|Center|Right|Char)" >

<!ENTITY % TblUnit  "(percent|pixels|points|picas|mm|cm|inches)" >

<!ENTITY % RProps
    'RowSep     %HBorder;  #IMPLIED
     VAlign     %VAlign;   #IMPLIED' >
<!ENTITY % CProps
    'ColSep     %VBorder;   #IMPLIED
     HAlign     %HAlign;    #IMPLIED
     AlignChr   CDATA       #IMPLIED
     CharPos    CDATA       #IMPLIED' >


<!-- This is the IBM-provided hook into IBMIDDOC -->
<!ELEMENT  SoftQuadTable    - - (%TblBody;) >
<!ATTLIST  SoftQuadTable
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "SoftQuadTable"
>


<!-- TblBody is the top-level table element -->
<!ELEMENT  %TblBody;        O O ((%TblCDfs;), (%TblRows;)) >
<!ATTLIST  %TblBody;
    TblWd            CDATA        #IMPLIED
    TblHt            CDATA        #IMPLIED
    TblUnits         %TblUnit;    #IMPLIED
    SQTable          NAME         #FIXED    "TBLBODY"
>

<!-- Table Column Definitions: -->
<!ELEMENT  %TblCDfs;        O O (%TblCDef;)+ >
<!ATTLIST  %TblCDfs;
    -- CProps with default values to apply to all columns --
    ColSep           %VBorder;    "VSingle"
```

```
    HAlign          %HAlign;     "Left"
    AlignChr        CDATA        #IMPLIED
    CharPos         CDATA        #IMPLIED
    -- default width for all cells in all columns --
    ColWd           CDATA        #IMPLIED
    TblUnits        %TblUnit;    #IMPLIED
    -- default top border for all cells in row 1 --
    TopSep          %HBorder;    "HSingle"
    SQTable         NAME         #FIXED    "TBLCDEFS"
>
<!-- there is exactly one TblCDef for each column in the table -->
<!ELEMENT  %TblCDef;        - O EMPTY
                                  -- Only exists to hold attributes --
>
<!ATTLIST  %TblCDef;
    %CProps;        -- overrides settings in TblCDfs --
    -- width for all cells in this column --
    ColWd           CDATA        #IMPLIED
    TblUnits        %TblUnit;    #IMPLIED
    -- top border for cell in row 1 --
    TopSep          %HBorder;    #IMPLIED  -- overrides TblCDfs defn --
    SQTable         NAME         #FIXED    "TBLCDEF"
>

<!-- Table Rows: -->
<!ELEMENT  %TblRows;        O O (%TblRow;)+ >
<!ATTLIST  %TblRows;
    -- RProps with default values to apply to all rows --
    RowSep          %HBorder;    "HSingle"
    VAlign          %VAlign;     "Top"
    -- default height of all rows --
    RowHt           CDATA        #IMPLIED
    TblUnits        %TblUnit;    #IMPLIED
    -- default left border for all cells in column 1 --
    LeftSep         %VBorder     "VSingle"
    SQTable         NAME         #FIXED    "TBLROWS"
>
<!ELEMENT  %TblRow;         - O (%TblCell;)+ >
<!ATTLIST  %TblRow;
    %RProps;    -- can override settings in TblRows --
    -- height of this row --
    RowHt           CDATA        #IMPLIED
    TblUnits        %TblUnit;    #IMPLIED
    -- left border for cell in column 1 --
    LeftSep         %VBorder     #IMPLIED
                                        -- overrides TblRows setting --
    SQTable         NAME         #FIXED    "TBLROW"
>

<!-- Table Cells -->
<!ELEMENT  %TblCell;        - O %CellMdl; >
<!ATTLIST  %TblCell;
    %RProps;        -- can override settings in TblRow --
    %CProps;        -- can override settings in TblCDef --
    -- cell positioning --
    ColStart        NUMBER       #IMPLIED
    ColSpan         NUMBER       "1"     -- span >= 1 --
    RowStart        NUMBER       #IMPLIED
```

```
    RowSpan          NUMBER       "1"     -- span >= 1 --
    SQTable          NAME         #FIXED    "TBLCELL"
>

]]>

<!--Long Quote ========================================================-->

<!ELEMENT  LongQuote        - O ((%InfoUnitPool;)*, Div*) >
<!ATTLIST  LongQuote
    BibID            NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Excerpt"
>

<!--Bridge ============================================================-->

<!ELEMENT  Bridge           - O (Title?, (%InfoUnitPool;)*) >
<!ATTLIST  Bridge
--  LinkEnds         NAMES        #IMPLIED  part of common attributes --
    %Common.Atts.NoLexMdl.NoHyTime;

    InfoMaster       NAME         #FIXED    "ConceptBridge"
    HyTime           NAME         #FIXED    "ilink"
    anchrole         NAMES        #FIXED    "link1 link2"
    LexModel         CDATA        #FIXED
                          'LinkEnds ((name | "#IMPLIED"), S,
                                          (name | "#IMPLIED")
                           &Common.LexModel;'
>

<!--Syntax Diagrams ===================================================-->

<!ELEMENT  Syntax           - -
                     (Title, Desc?, Set*, RepSep*,
                      (Group | Range |
                       FragmentRef | Fragment | SyntaxBlock |
                       SyntaxNote)*
                     )
                                                -- Syntax diagram --
>
<!ATTLIST  Syntax
    CompLang         CDATA        #IMPLIED
       --Defines the computer language to which the definition applies--
    %Common.Atts;
>

<!ELEMENT  SyntaxBlock      - O
                     (Title, Desc?, Set*, RepSep*,
                      (Group | Range | FragmentRef | Fragment |
                       SyntaxNote)*
                     )                             -- Syntax block --
>
<!ATTLIST  SyntaxBlock
    %Common.Atts;
>
```

```
<!ELEMENT  Set              - - (Title?, (setitem | setref | setspec)*) >
<!ATTLIST  Set
    ID              NAME         #REQUIRED
    SetOrder        (ordered | unordered)
                                 unordered
    %Common.Atts.NoID;
>

<!ELEMENT  SetItem          - O (%DataPool;)* >
<!ATTLIST  SetItem
    %Common.Atts;
>

<!ELEMENT  SetRef           - O EMPTY               -- Set Reference -->
<!ATTLIST  SetRef
    ID              NAME         #IMPLIED
    SetID           NAME         #REQUIRED    -- Must be ID of a SET --
    %Common.Atts.Base;

    HyNames         CDATA        #FIXED    '&Common.HyNames;'
    LexModel        CDATA        #FIXED    '&Common.LexModel;'
    RefType         CDATA        #FIXED
                         "SETID SET &Common.RefType;"
>

<!ELEMENT  SetSpec          - O (%DataPool;)*    -- Set Specification -->
<!ATTLIST  SetSpec
    SetNotation     CDATA        #IMPLIED      -- Default is "prose" --
          -- Notation is not required to be an SGML-declared notation --
    %Common.Atts;
>

<!ENTITY % GroupElem
    "Group | Range | FragmentRef | Kwd | Var | Delim | Oper | Sep |
     SyntaxNote"
>

<!ELEMENT  Group            - - (Title?, Desc?, (%GroupElem;)*) >
<!ATTLIST  Group
    Repeat          CDATA        #IMPLIED
    RepID           NAME         #IMPLIED       -- Must be REPSEP ID --
    RepSep          CDATA        #IMPLIED  -- Repeat separator chars --
    OptReq          (Opt | Req| Def)  Req
    ChoiceSeq       (Seq | Choice)
                                 Seq
    GroupOrder      (Ordered | Unordered)
                                 Ordered
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel        CDATA        #FIXED
                         "REPEAT ('YES' | 'NO' | NUMBER | RCDATA)
                          &Common.LexModel;"
    RefType         CDATA        #FIXED
                         "RepID RepSep &Common.RefType;"
>

<!ELEMENT  Fragment         - -
                    (Title, Desc?, RepSep*,
```

```
                     (Group | Range | FragmentRef | SyntaxNote)*)
>
<!ATTLIST  Fragment
    %Common.Atts;
>

<!ELEMENT  FragmentRef          - O (Title)     -- Fragment Reference -->
<!ATTLIST  FragmentRef
    FragID           NAME         #CONREF      -- Must be Fragment ID --
    Repeat           CDATA        #IMPLIED           -- Default is NO --
    RepID            NAME         #IMPLIED       -- Must be REPSEP ID --
    RepSep           CDATA        #IMPLIED  -- Repeat separator chars --
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                          "REPEAT ('YES' | 'NO' | number | rcdata)
                           &Common.LexModel;"
    RefType          CDATA        #FIXED
                          "RepID RepSep FragID Fragment &Common.RefType;"
    TargetType       CDATA        #FIXED    "Fragment"
    InfoMaster       NAME         #FIXED    "SymbolicReference"
>

<!ELEMENT  Range            - O (Title, RangeStart?, RangeEnd?) >
<!ATTLIST  Range
    Repeat           CDATA        #IMPLIED
    RepID            NAME         #IMPLIED -- Must be REPSEP ID --
    SetID            NAME         #IMPLIED -- Must be SET ID --
    RepSep           CDATA        #IMPLIED -- Repeat separator chars --
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                          "REPEAT ('YES' | 'NO' | number | rcdata)
                           &Common.LexModel;"
    RefType          CDATA        #FIXED
                          "REPID REPSEP SETID SET &Common.RefType;"
>

<!ELEMENT  (RangeStart | RangeEnd)
                            - O (%DataPool;)* >
<!ATTLIST  (RangeStart | RangeEnd)
    %Common.Atts;
>

<!ELEMENT  Var              - O (%DataPool;)*      -- Syntax variable -->
<!ATTLIST  Var
    SetID            NAME         #IMPLIED          -- Must be SET ID --
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts.NoRefType;

    RefType          CDATA        #FIXED
                          "SETID SET &Common.RefType;"
>

<!ELEMENT  Kwd              - O (%DataPool;)*       -- Syntax keyword -->
<!ATTLIST  Kwd
```

```
    Abbreviations    CDATA        #IMPLIED
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;
>

<!ELEMENT  Oper             - O (%DataPool;)*      -- Syntax Operator -->
<!ATTLIST  Oper
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;
>

<!ELEMENT  SyntaxNote       - - (FNBody)               -- Syntax note -->
<!ATTLIST  SyntaxNote
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!ELEMENT  RepSep           - - (Title, (%GroupElem;)*)
                             -- Separator for repeated syntax tokens --
>
<!ATTLIST  RepSep
    OptReq           (Opt | Req| Def)  Req
    CHOICESEQ        (Seq | Choice)
                                 Seq
    GROUPORDER       (ordered | unordered)
                                 Ordered
    %Common.Atts;
>

<!ELEMENT  Sep              - O (%DataPool;)*     -- Syntax Separator -->
<!ATTLIST  Sep
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;
>

<!ELEMENT  Delim            - O (%DataPool;)*     -- Syntax Delimiter -->
<!ATTLIST  Delim
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;
>

<!--Modular Information ================================================-->

<!ELEMENT  ModularInformation
                              - - (InfoModule+)
>
<!ATTLIST  ModularInformation
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleList"
>

<!ELEMENT  InfoModule       - O (ModuleName+, ModuleDesc?, ModuleItem+)
```

```
                                              -- Information Module --
>
<!ATTLIST  InfoModule
    Class            NAME         #REQUIRED         -- author-defined --
    %Common.Atts.NoClass;

    InfoMaster       NAME         #FIXED    "InformationModule"
>

<!ELEMENT  ModuleName       - O (%DataPool;)* >
<!ATTLIST  ModuleName
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleName"
>

<!ELEMENT  ModuleDesc       - O (%InfoPool;)*   -- Module Description -->
<!ATTLIST  ModuleDesc
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Description"
>

<!ELEMENT  ModuleItem       - O (%InfoPool;)*          -- Module Item -->
<!ATTLIST  ModuleItem
    Class            CDATA        #REQUIRED
    -- Valid values defined on ModularInfoDef element --
    %Common.Atts.NoClass

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ModuleItem"
>

<!--Language Element Reference Section ================================-->

<!ELEMENT  LERS             - - (LE+)
                                -- Language Element Reference Section --
>
<!ATTLIST  LERS
    %lersdef.atts;
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleList"
    ApplicationClass NAME         #FIXED    "ComputerLanguage"
>

<!ELEMENT  LE               - O (LEName, LEDesc?, LEDI+)
                                              -- Language Element --
>
<!ATTLIST  LE
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
    ApplicationClass NAME         #FIXED    "ComputerLanguage"
>

<!ELEMENT  LEName           - O (%DataPool;)*
```

```
                                          -- Language Element Name --
>
<!ATTLIST  LEName
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleName"
>

<!ELEMENT  LEDesc           - O (%DataPool;)*
                                    -- Language element description --
>
<!ATTLIST  LEDesc
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescription"
>

<!ELEMENT  LEDI             - O ((%InfoUnitPool;)* | Div+)
                                 -- Language Element Description Item --
>
<!ATTLIST  LEDI
    Class            (Auth |
                      Comments |
                      Context |
                      Defaults |
                      Errcond |
                      Examples |
                      Flags |
                      Format |
                      Intrep |
                      Sep |
                      Messages |
                      Other |
                      Parms |
                      Process |
                      Purpose |
                      Restrict |
                      Results |
                      Retcodes |
                      Sysenv |
                      Usage |
                      Ver )
                                 #REQUIRED
    %Common.Atts.NoClass;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "LanguageElementItem"
>

<!--MAN Pages ============================================================-->

<!ENTITY % CommonMan-ManPage
    "ManName, ManSynopsis, ManDesc, ManUsage, ManSeeAlso, ManNotices"
                                          -- temporary place holder --
>

<!ELEMENT  ManPage          - - (%CommonMan-ManPage;) >
<!ATTLIST  ManPage
```

```
    ManIndex         CDATA        #REQUIRED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
>

<!ELEMENT  ManName          - O (%DataPool;)* >
<!ATTLIST  ManName
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANName"
>

<!ELEMENT  ManSynopsis      - O (%InfoPool;)* >
<!ATTLIST  ManSynopsis
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANSynopsis"
>

<!ELEMENT  ManDesc          - O (%InfoPool;)* >
<!ATTLIST  ManDesc
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANDesc"
>

<!ELEMENT  ManUsage         - O (%InfoPool;)* >
<!ATTLIST  ManUsage
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANUsage"
>

<!ELEMENT  ManSeeAlso       - O (%DataPool;)* >
<!ATTLIST  ManSeeAlso
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANSeeAlso"
>

<!ELEMENT  ManNotices       - O (%InfoUnitPool;)* >
<!ATTLIST  ManNotices
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MANNotices"
>

<!--Messages and Codes List ========================================-->

<!ELEMENT  MessageList      - - (MessageItemDef*, Message+) >
<!ATTLIST  MessageList
```

```
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  MessageItemDef   - O (Title, Desc?)
 -- Title element defines default title for a given MessageItem class --
                                        -- Message Item Definition --
>
<!ATTLIST  MessageItemDef
    Class            CDATA        #REQUIRED
    %Common.Atts.NoClass

    InfoMaster       NAME         #FIXED    "PropertyDefinition"
>

<!ELEMENT  Message          - O (((CodeNumber* | MessageNumber*),
                                  MessageText?)*,
                                 MessageItem*)
                                                -- Message Module --
>
<!ATTLIST  Message
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
>

<!ELEMENT  CodeNumber       - O (%DataPool;)* >
<!ATTLIST  CodeNumber
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "CodeNumber"
>

<!ELEMENT  MessageNumber    - O (%DataPool;)* >
<!ATTLIST  MessageNumber
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MessageNumber"
>

<!ELEMENT  MessageText      - O (%DataPool;)* >
<!ATTLIST  MessageText
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MessageText"
>

<!ELEMENT  MessageItem      - O ((%InfoUnitPool;)*, Div)
                                        -- Message Description Item --
>
<!ATTLIST  MessageItem
    Class            CDATA        #REQUIRED
                    -- Valid values defined on MessageItemDef element --
    %Common.Atts.NoClass;
```

```
    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "MessageItem"
>

<!--Procedures ========================================================-->

<!ELEMENT  Procedure        - -
                     (Title, Desc?, ProcEntry, ProcStep+, ProcExit)
>
<!ATTLIST  Procedure
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "InformationModule"
    ApplicationClass NAME         #FIXED    "Procedure"
>

<!ELEMENT  ProcEntry        - O (Desc, Prerequisites?, RelatedProcs?)
                                  -- Procedure entry information --
>
<!ATTLIST  ProcEntry
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ProcEntry"
>

<!ELEMENT  (Prerequisites | RelatedProcs)
                           - O (#PCDATA | ProcRef)*
>
<!ATTLIST  (Prerequisites | RelatedProcs)
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                          "#CONTENT (NAME | #GI)*
                           &Common.LexModel;"
    RefType          CDATA        #FIXED
                          "#CONTENT (Procedure | LinkDesc)
                           &Common.RefType;"
>

<!ELEMENT  ProcStep         - O
                     (Title, ReferenceKeys?, Desc, StepNotes?,
                      DecisionPoint?)
                                              -- Procedure step --
>
<!ATTLIST  ProcStep
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ProcStep"
>

<!ELEMENT  ReferenceKeys    - O (#PCDATA | RefKey)* >
<!ATTLIST  ReferenceKeys
    %Common.Atts.NoLexMdl;

    LexModel         CDATA        #FIXED
```

```
                        "#CONTENT (NAME | #GI)
                         &Common.LexModel;"
>

<!ELEMENT  StepNotes        - O (Title?, (LI | LIBlock | Bridge)+) >
<!ATTLIST  StepNotes
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "SinglePartList"
    InfoMasterSemanticProperty
                     NAME         #FIXED    "Enumeration"
>

<!ELEMENT  ProcExit         - O
                     (Desc, ExpectedResults?, RecoveryProc?,
                      DecisionPoint?)
                                    -- Procedure exit information --
>
<!ATTLIST  ProcExit
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleDescriptionItem"
    ApplicationClass NAME         #FIXED    "ProcExit"
>

<!ELEMENT  (ExpectedResults | RecoveryProc)
                            - O (%InfoPool;)*
>
<!ATTLIST  (ExpectedResults | RecoveryProc)
    %Common.Atts;
>

<!ELEMENT  DecisionPoint    - - (Condition, Action)+ >
<!ATTLIST  DecisionPoint
    %Common.Atts;
>

<!ELEMENT  Condition        - O (%DataPool;)* >

<!ELEMENT  Action           - O (%DataPool;)* >

<!ATTLIST  (Condition | Action)
    %Common.Atts;
>

<!ELEMENT  ProcRef          - - (%DataPool;)*  -- Procedure Reference -->
<!ATTLIST  ProcRef
    %Common.Atts.NoLexMdl.NoRefType;

    LexModel         CDATA        #FIXED
                          "#CONTENT (NAME | #GI)*
                           &Common.LexModel;"
    RefType          CDATA        #FIXED
                          "#CONTENT (Procedure | LinkDesc)
                           &Common.RefType;"
    InfoMaster       NAME         #FIXED    "LogicalReference"
>
```

```
<!--Bibliographic Reference List ====================================-->

<!ELEMENT  BibList          - O (Title?,
                                 (nmquery |
                                  (%BibRecord; | Citation | %LibRecord;)*
                                 ))
                                                  -- Bibligraphy List --
>
<!ATTLIST  BibList
    EntryType        (Document | Library)
                                 Document
    Form             (Full | Title | TitleNumber | Normal)
                                 Normal
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ModuleList"
>

<!--Marked Notes ===================================================-->

<!ELEMENT  MarkedNote       - - (%InfoPool;)* >
<!ATTLIST  MarkedNote
    Class            NAMES        #REQUIRED
    Action           NAMES        #REQUIRED
    MarkIDs          NAMES        #REQUIRED   -- IDs of MARK elements --
    Item             CDATA        #IMPLIED
    %Common.Atts.NoRefType.NoClass;

    RefType          CDATA        #FIXED
                          "MarkIDs Mark &Common.RefType"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!ELEMENT  MarkDesc         - - (MarkClass | MarkAction)* >
<!ATTLIST  MarkDesc
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ContainingDivision"
>

<!ELEMENT  MarkClass        - O (%DataPool;)* >
<!ATTLIST  MarkClass
    Name             NAME         #REQUIRED
    %Common.Atts;
>

<!ELEMENT  MarkAction       - O (%DataPool;)* >
<!ATTLIST  MarkAction
    name             NAME         #REQUIRED
    %Common.Atts;
>

<!ELEMENT  Mark             - O (%DataPool;)* >
<!ATTLIST  Mark
    ID               NAME         #REQUIRED
    run              (yes|no)     #REQUIRED
```

```
    %Common.Atts.NoID;
>

<!--Annotation Elements ================================================-->

<!ELEMENT  FN              - - (Title?, FNBody)          -- FootNote -->
<!ATTLIST  FN
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!ELEMENT  FNBody          O O (%InfoPool;)*       -- Foot Note Body -->
<!ATTLIST  FNBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ParagraphBody"
>

<!ELEMENT  Annot           - - (Title, AnnotBody)      -- Annotation -->
<!ATTLIST  Annot
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
>

<!ELEMENT  AnnotBody       O O (%InfoPool;)*      -- Annotation Body -->
<!ATTLIST  AnnotBody
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ParagraphBody"
    -- Its semantic class is inherited from its parent: Annot --
>

<!--Common MetaInformation Subelements ================================-->

<!ELEMENT  Title           O O (ShortTitle?, TitleBody) >
<!ATTLIST  Title
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Title"
>

<!ELEMENT  ShortTitle      - - (%DataPool;)* >
<!ATTLIST  ShortTitle
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ShortTitle"
>

<!ELEMENT  TitleBody       O O (%DataPool;)* >
<!ATTLIST  TitleBody
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "TitleBody"
>

<!ELEMENT  Caption          - O (%DataPool;)* >
<!ATTLIST  Caption
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Caption"
>

<!ELEMENT  Desc             - O (%InfoPool;)*      -- Description -->
<!ATTLIST  Desc
    %Common.Atts;
    InfoMaster       NAME         #FIXED    "Description"
>

<!--Artwork and Other Special Data ==================================-->

<!ELEMENT  CGraphic         - O RCDATA           -- Character graphic -->
<!ATTLIST  CGraphic
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    Notation         (CGraphic)
                                  #FIXED    "CGraphic"
    ApplicationClass NAME         #FIXED    "CharacterGraphic"
>

<!ELEMENT  Lines            - - (%DataPool;)*   -- Line-specific data -->
<!ATTLIST  Lines
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    Notation         (LineSpecific)
                                  #FIXED    "LineSpecific"
    ApplicationClass NAME         #FIXED    "Lines"
>


<!ENTITY % LitData.Notations
    "%ComputerLanguage.Notations; | %TextLanguage.Notations; |
     LineSpecific"
>

<!ELEMENT  Litdata          - O CDATA                -- Literal data -->
<!ATTLIST  Litdata
    Object           ENTITY       #CONREF
    Notation         (%LitData.Notations;)
                                  #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    ApplicationClass NAME         #FIXED    "LiteralData"
>

<!ELEMENT  MultiMediaObject - O
```

```
                    ((ObjectContainer | Link | Xref)*, TextAlt)
>
<!ATTLIST  MultiMediaObject
    Object          ENTITY       #IMPLIED
    %Common.Atts;

    InfoMaster      NAME         #FIXED     "MultiMediaContainer"
>

<!ELEMENT  TextAlt          - O (%InfoPool;)*
>
<!ATTLIST  TextAlt
    %Common.Atts;

    InfoMaster      NAME         #FIXED     "TextAlternative"
>

<!-- These notations may be included in line within an SGML document.
     Other data which may not be included in line are referenced
     with a data entity reference, which identifies its notation
     on the entity declaration, not on the container element.
     Care must be taken when including graphics data
     inline to avoid misinterpretation of graphics commands as
     SGML markup. -->
<!ENTITY % ObjectContainer.Notations
    "EPS | CGMChar | %Math.Notations; |
     %ComputerLanguage.Notations; | %TextLanguage.Notations; |
     LineSpecific | LiteralData"
>

<!ELEMENT  ObjectContainer  - O (%DataPool;)*
                                          -- data object container --
>
<!ATTLIST  ObjectContainer
    Object           ENTITY       #CONREF
    Notation         (%ObjectContainer.Notations;)
                                  LiteralData
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    ApplicationClass NAME         #FIXED    "NonTextObject"
>

<!-- DTL notation data is referenced using the Object attribute -->
<!ENTITY % Screen.Notations "LineSpecific | CGraphic" >

<!ELEMENT  Screen           - - (%DataPool;)* >
<!ATTLIST  Screen
    Object           ENTITY       #CONREF
    Notation         (%Screen.Notations;)
                                  LineSpecific
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    ApplicationClass NAME         #FIXED    "Screen"
>

<!ELEMENT  XMP              - O (%DataPool;)*
```

```
                                -- computer input and output example --
>
<!ATTLIST  XMP
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    Notation         (LineSpecific)
                                  #FIXED    "LineSpecific"
    ApplicationClass NAME         #FIXED    "ComputerExample"
>


<!--====================================================================
    Data Pool Elements
    ====================================================================-->

<!--General Phrases ================================================-->

<!-- See Bibliography information in Prolog for BibRecord element -->

<!ELEMENT  Citation         - O (%BibRecord; | %LibRecord;) >
<!ATTLIST  Citation
    BibID            NAME         #CONREF             -- BibRecord ID --
    Form             (Full | Title | DocNum | Normal)
                                  Normal
    ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    %Common.Atts.Base;

    InfoMaster       NAME         #FIXED    "phrase"
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED
                          "BibID (&BibRecord;)
                           &Common.RefType;"
>

<!ELEMENT  Phrase           - - (%DataPool;)*       -- Generic phrase -->
<!ATTLIST  Phrase
    SearchClass      NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
>

<!ELEMENT  Quote            - O (%DataPool;)* >
<!ATTLIST  Quote
    BibID            NAME         #IMPLIED
    %Common.Atts.NoRefType;

    RefType          CDATA        #FIXED
                          "BibID (&BibRecord;)
                           &Common.RefType;"
    InfoMaster       NAME         #FIXED    "Excerpt"
>
```

```
<!ELEMENT  DefinedTerm      O O (%DataPool;)* >
<!ATTLIST  DefinedTerm
    SearchClass      NAME         #IMPLIED
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "DefinedTerm"
    Notation         (Term)
                                  #FIXED    "Term"
>

<!--Technical Phrases ================================================-->

<!ELEMENT  APL              - - (#PCDATA)          -- APL data phrase -->
<!ATTLIST  APL
    Object           ENTITY       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    Notation         (APL)
                                  #FIXED    "APL"
    ApplicationClass NAME         #FIXED    "APLData"
>

<!ELEMENT  Bin              - - (#PCDATA)       -- Binary data phrase -->
<!ATTLIST  Bin
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "BinaryData"
    LexModel         CDATA        #FIXED
                          "#CONTENT ('0' | '1' | ' ')*
                           &Common.LexModel;"
>

<!ELEMENT  Char             - - (#PCDATA)
                                    -- literal character data phrase --
>
<!ATTLIST  Char
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "CharacterData"
>

<!ELEMENT  Date             - O (#PCDATA) >
<!ATTLIST  Date
    Auto             (Auto)       #CONREF
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Paragraph"
>

<!ELEMENT  Decimal          - - (#PCDATA)      -- decimal data phrase -->
<!ATTLIST  Decimal
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "DecimalData"
```

```
    LexModel         CDATA        #FIXED
                          "#CONTENT (digits | '.' | ',' | ' ')*
                           &Common.LexModel;"
>

<!ELEMENT  Formula          - - CDATA >
<!ATTLIST  Formula
    Object           ENTITY       #CONREF
    Notation         (%Math.Notations)
                                  #IMPLIED
                                       -- Spec'd if Object not spec'd --
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "ObjectContainer"
    ApplicationClass NAME         #FIXED    "Mathematics"
>

<!ELEMENT  Hex              - - (#PCDATA)  -- hexadecimal data phrase -->
<!ATTLIST  Hex
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "HexidecimalData"
    LexModel         CDATA        #FIXED
                          "#CONTENT
                              (digits | 'A'|'B'|'C'|'D'|'E'|'F' | ' ')*
                           &Common.LexModel;"
>

<!ELEMENT  MD               - - (%DataPool;)*      -- Marked deletion -->

<!ATTLIST  MD

--  Revision         NAME         #REQUIRED --
--  It would be desirable for Revision to be required but since it is --
--  already specified in %Common.Atts;, it is not coded here. --

    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "MarkedDeletion"

>

<!ELEMENT  MV               - - (%DataPool;)*     -- Message Variable -->
<!ATTLIST  MV
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "MessageVariable"
>

<!ELEMENT  Number           - - (#PCDATA) >
<!ATTLIST  Number
    Base             NUMBER       #REQUIRED
    %Common.Atts;
```

```
    InfoMaster       NAME         #FIXED    "Phrase"
>

<!ELEMENT  Octal            - - (#PCDATA)        -- Octal data phrase -->
<!ATTLIST  Octal
    %Common.Atts.NoLexMdl;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "OctalData"
    LexModel         CDATA        #FIXED
                          "('0'|'1'|'2'|'3'|'4'|'5'|'6'|'7' | ' ')*
                           &Common.LexModel;"
>

<!ELEMENT  RefKey           - - (RefKeyBody, desc?, NotLoc?)
                                                 -- Reference Key --
>
<!ATTLIST  RefKey
--  LinkEnds         NAMES        #IMPLIED  part of common attributes --
    %Common.Atts.NoLexMdl.NoHyTime;

    anchrole         CDATA        #FIXED    "textanchor graphicanchor"
    LexModel         CDATA        #FIXED
                          "LinkEnds ( (idref | '#IMPLIED'),
                                      (idref | '#IMPLIED'))
                           &Common.LexModel;"
    HyTime           NAME         #FIXED    "ilink"
                                       -- HyTime independent link form --
>

<!ELEMENT  RefKeyBody       O O (%DataPool;)*   -- Reference Key Body -->
<!ATTLIST  RefKeyBody
    %Common.Atts;
>

<!ELEMENT  SyntaxPhrase     - - (Desc?, Set*, RepSep*, (%GroupElem;)*)
                                                 -- Syntax phrase --
>
<!ATTLIST  SyntaxPhrase
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "SyntaxPhrase"
>

<!ELEMENT  PK               - - (%DataPool;)*
        -- Parameter keyword == <syntaxphrase><kwd>...</syntaxphrase> --
>
<!ATTLIST  PK
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "SyntaxPhrase"
>

<!ELEMENT  PV               - - (%DataPool;)*
```

```
       -- Parameter variable == <syntaxphrase><var>...</syntaxphrase> --
>
<!ATTLIST  PV
    SetID            NAME         #IMPLIED          -- Must be SET ID --
    OptReq           (Opt | Req| Def)  Req
    %Common.Atts.NoRefType;
    RefType          CDATA        #FIXED
                          "SETID SET &Common.RefType;"

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "SyntaxPhrase"
>


<!ELEMENT  XPH              - - (%DataPool;)*
                            -- Computer input or output example phrase --
>
<!ATTLIST  XPH
    %Common.Atts;

    InfoMaster       NAME         #FIXED    "Phrase"
    ApplicationClass NAME         #FIXED    "ExamplePhrase"
>

<!--Linking Phrases ================================================-->

<!ELEMENT  Link             - - (%DataPool;)*
                                       -- Explicit hyperlink phrase --
>
<!ATTLIST  Link
    LinkEnd          NAME         #REQUIRED
    LinkPhrase       (Auto)       #CONREF
    ID               NAME         #IMPLIED
    ContentRef       NAME         #CONREF
    XRefText         CDATA        #IMPLIED
    InfoType         CDATA        #IMPLIED
    %Hyperlink.Attributes;
    %Property.Attributes;

    InfoMaster       NAME         #FIXED    "HyperLink"
    -- HyTime attributes: --
    HyTime           NAME         #FIXED    "ilink"
    HyNames          CDATA        #FIXED
                          "linkend LinkEnd &Common.HyNames;"
    RefType          CDATA        #FIXED
                          "#ALL #ANY &Common.RefType;"
                       -- No REFLEVEL restrictions (see 10744 6.5.3) --
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    refrange         CDATA        #FIXED    "#ALL I"   -- indirection --
>

<!ELEMENT  XRef             - O EMPTY      -- Cross Reference Element -->
<!ATTLIST  XRef
    ID               NAME         #IMPLIED
    RefID            NAME         #REQUIRED
    Form             (normal | full | text | number | location |
                      textnum | locnum | textloc |
                      numtext | numloc | loctext)
```

```
                              normal
    %Common.Atts.Base;

    InfoMaster       NAME         #FIXED    "CrossReference"
    -- HyTime attributes: --
    HyTime           NAME         #FIXED    "ilink"
    reftype          CDATA        #FIXED
                          "#ALL #ANY &Common.RefType;"
                        -- No REFLEVEL restrictions (see 10744 6.5.3) --
    refrange         CDATA        #FIXED    "#ALL I"   -- indirection --
    LexModel         CDATA        #FIXED
                          "RefID IDREFS &Common.LexModel;"
    HyNames          CDATA        #FIXED
                          "linkend RefID &Common.HyNames;"
>

<!--Indexing Phrases ================================================-->

<!ELEMENT  IndexTerm        O O (%DataPool;)*
                                    -- index term PCDATA container --
>
<!ATTLIST  IndexTerm
    %Common.Atts;
>

<!ELEMENT  I1               - O (IndexTerm+, I2*)
                                     -- first-level index entry --
>
<!ATTLIST  I1
    SeeText          CDATA        #IMPLIED
    SeeID            NAMES        #IMPLIED
    SortKey          CDATA        #IMPLIED
    Primary          (Primary)    #IMPLIED          -- Primary entry? --
    %Common.Atts.NoRefType;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
    RefType          CDATA        #FIXED
                          "SeeAlso I1 &Common.RefType;"
>

<!ELEMENT  I2               - O (IndexTerm, I3*)
                                    -- second-level index entry --
>
<!ATTLIST  I2
    I1ID             NAME         #IMPLIED               -- I1 parent --
    Primary          (Primary)    #IMPLIED          -- Primary entry? --
    SeeID            CDATA        #IMPLIED
    SeeAlso          NAMES        #IMPLIED
    SortKey          CDATA        #IMPLIED
    %Common.Atts.NoRefType;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
    RefType          CDATA        #FIXED
                          "I1ID I1 SeeAlso I1
                           &Common.RefType;"
>
```

```
<!ELEMENT  I3              - O (IndexTerm)
                                        -- third-level index entry --
>
<!ATTLIST  I3
    I2ID             NAME         #IMPLIED               -- I2 parent --
    Primary          (Primary)    #IMPLIED          -- Primary entry? --
    SortKey          CDATA        #IMPLIED
    %Common.Atts.NoRefType;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
    RefType          CDATA        #FIXED
                          "I2ID I2
                           &Common.RefType;"
>

<!ELEMENT  IRef             - O EMPTY
                     -- Index reference.  Refers to an I1, I2, or I3. --
>
<!ATTLIST  IRef
    ID               NAME         #IMPLIED
    RefIDs           NAMES        #REQUIRED
    Primary          (Primary)    #IMPLIED          -- Primary entry? --
    %Common.Atts.Base;

    InfoMasterSemanticClass
                     NAME         #FIXED    "Annotation"
    HyNames          CDATA        #FIXED    '&Common.HyNames;'
    LexModel         CDATA        #FIXED    '&Common.LexModel;'
    RefType          CDATA        #FIXED
                          "RefIDs (I1 | I2 | I3) &Common.RefType;"
>

<!--================================================================
    Character entity declarations for BookMaster processing
    ================================================================-->

<!ENTITY   ctq              SDATA    "IBMIDDOC: ctq"
                                        --  close typographic quote  --
>
<!ENTITY   otq              SDATA    "IBMIDDOC: otq"
                                         --  open typographic quote  --
>

<!ENTITY % UseBMSymbols     "ignore" >
<!ENTITY % UseISOCharEnts   "include" >

<!ENTITY % BookMasterSymbols PUBLIC
    "+//ISBN 0-933186::IBM//ENTITIES Entities for BookMaster Symbols//EN"
>

<!ENTITY % ISOCharEntities  PUBLIC
    "+//ISBN 0-933186::IBM//ENTITIES ISO Symbol Sets for BookMaster//EN"
>
```

```
<![ %UseBMSymbols; [
%BookMasterSymbols;
]]>

<![ %UseISOCharEnts; [
%ISOCharEntities;
]]>


<!--====================================================================
    Default entity definition
    ====================================================================-->

<!entity   #default         SDATA  "IBMIDDOC: {Undefined entity}" >

<!--====================================================================
    HyTime Declarations
    ====================================================================-->
<?HyTime VERSION "ISO/IEC 10744:1992" HYQCNT=32>
<?HyTime MODULE base>
<?HyTime MODULE locs>
<?HyTime MODULE links>
```

Glossary

Since IBM ID Doc is an SGML application, SGML's terminology will be used whenever it applies and avoided when it does not. For the definition of an SGML term, refer to *ISO 8879:1986 — Standard Generalized Markup Language* or to *The SGML Handbook*.

application. An application is a collection of information class definitions. In SGML terms, an application is defined by a DTD which defines what information structures (elements) can and cannot be found in documents which conform to the application.

application domain. An application domain is a class of documents which is characterized by unique set information classes. They may also share the same application definition since this is a codification of the information classes which comprise the application domain.

architecture. An architecture is a definition or processing scheme which allows a single system to process documents from multiple application domains which are defined by different applications. Common processing is shared by all these domains or applications by abstracting to the information classes they share and processing each information class with the same code.

architectural forms. Architectural forms are a concept introduced by the HyTime standard. These forms are meta-element declarations and meta-attribute declarations. That means that they guide the writer of DTDs in writing DTDs by indicating how elements and attributes must be coded. If a DTD follows the rules indicated by these forms, it may be processed by systems designed to process data which conforms to the forms. That is the purpose of the definition of these forms, to codify limits so that processing systems may process data.

These architectural forms are the codification of the rules of information classes that characterize applications and therefore, application domains.

BookMaster. BookMaster is a product sold by IBM which defines a markup language for documents and then processes documents which use this language using Document Composition Facility (DCF). As used in this document, BookMaster may also mean only the markup language definition (exclusive of the processsing system).

character entity. The term character entity is used to describe an entity which represents a single character. Typically, this character is not representable in the document character set or at least cannot be entered on the keyboard that the author is using to edit the document. The replacement text of a character entity is often SDATA, since representation of a character not in the document character set is inherently system-dependent.

control word. This is the term used to describe the character sequences used by DCF to control its processing of document data. There are a large number of these control sequences defined by DCF. Typically, these control sequences start with a period which must be found in column one of an input line in the document data being processed.

cross-reference. A cross-reference is a hyperlink between two locations. It is unidirectional and typically causes generated text to be inserted by processing programs at the source point of the hyperlink. This generated text gives information about the information found at the other end of the hyperlink.

DCF. See “Document Composition Facility”.

document character set. The document character set is the collection

Document Composition Facility. Document Composition Facility (DCF) is a program product sold by IBM which takes document data marked up with special controls (called “control words”) and with generic markup called GML to produce hardcopy for a variety of printers.

Document Style Semantics Specification Language. This is the product of some work in progress in the international standards community which will result in an international standard. The standard will codify document formatting semantics. It will do this in the context of an architecture which can then be used to codify many other document processing semantics. Of particular interest is its processing model, which is used in the InfoMaster and IBM ID Doc processing models.

DSSSL. See “Document Style Semantics Specification Language”.

document type declaration. The document type declaration (DTD) is the SGML construct where all the elements, attributes and other markup constructs that are used in the document are declared. It is part of the prolog of an SGML document.

DTD. See “document type declaration”.

element. Element is an SGML concept. It contains the information associated with an information class. It may be further subdivided into other elements. It consists of a start tag (including any attributes), an end tag and content.

© Copyright IBM Corp. 1992, 1993

element type. Element type is the term used to refer to an element's definition, its name, its content model, its attributes, rather than a particular instance or usage of the element in a document. GD is an element type. This paragraph is found within an instance of a GD.

entity. An entity is an SGML construct which represents several ideas:

- It may be like a macro in the DTD, this type of entity is called a “parameter entity”.
- It may be like a GML symbol, with references replaced by a character string.
- It may define an external file containing SGML data. In this case, the reference is like the .im DCF control word, which references external GML data (only in SGML it is used to reference SGML data).
- It may define an external non-SGML file. In this case, the reference is like the .SI or .PO DCF control words or the NAME attribute of the ARTWORK BookMaster tag.

Entities are always defined in the prolog of the document.

entity reference. An entity reference is an SGML construct which is replaced by the information defined for the entity. The nature of the information which replaces the reference is dependent on the type of entity was defined.

formal public identifier. An FPI is a form of an SGML public identifier which conforms to the rules for formal public identifiers in the SGML standard. It is a system-independent way of refering to files from within an SGML document. Many SGML systems provide a means for converting these identifiers into a system identifier that the system processing the data can use to access data directly.

FPI. See “formal public identfier”.

generic identifier. A generic identifier is the name of an element type.

Generic Markup Language. GML is a generic markup scheme supported by Document Composition Facility. This scheme includes a tag (which may be a start or end tag), either of which may include “attributes” which further describe the data being “tagged”. While the delimiters used may be changed, the most commonly used GML delimiters use the character string ":" to open a start tag, ":e" to open an end tag and a period to close a tag. Each of these tags are then mapped, using DCF facilities for this purpose, to collections of DCF control words which are called macros. GML therefore has the capability (that SGML also has) of isolating the processing (the macros) from the data itself, which contains only these generic tags. Symbols are also supported. References to symbols in data are opened with an ampersand and closed with a period.

GML. See “Generic Markup Language”.

hyperlink. A hyperlink is a generic term for document structures which represent connections with locations in the same or other documents.

HyTime. HyTime is the short name for the “International Standard ISO 10744 — Hypermedia/Time-based Structuring Language”. This is an international standard which codifies the interchange of hypertext/hypermedia documents and the semantics represented in those documents. The standard is based on SGML but has the capability of representing connections or links between virtually any kind of data.

IBM ID Doc. IBM ID Doc is a new, SGML-based, application which is the follow-on to the BookMaster markup language. It supports a much more general processing model so that it may be supported by many different products on different platforms. It is also intended to be extendable to support multiple application domains. It is based on the InfoMaster architecture.

InfoMaster. InfoMaster is an architecture which was designed specifically to support IBM ID Doc and to meet IBM's needs in processing other industry standard markup applications. It consists of a number of architectural forms which represent information classes as well as some special semantics which InfoMaster processing systems must process to support the processing of documents which conform to the InfoMaster architecture.

information class. An information class is a set of information that represents a particular organization of data, like an ordered list, a table, a paragraph, etc.. An information class is represented by one or more elements which identify information which are instances of these classes.

information development. Information development is the name for organizations (at least in IBM) which create the documentation and other product information used by the customer.

ISIL. This is an abbreviation for the name of the internal publishing support that evolved into the IBM product known as BookMaster.

locator. In InfoMaster and IBM ID Doc, a generic term for any indicator of where a particular piece of information is located within a document. Typical locators are page numbers and panel identifiers. Locators are normally generated automatically by the processing

system and may be artifacts of the rendering process, as in the case of page numbers.

markup application. See “application”.

PI. See “processing instruction”.

processing instruction. Processing instructions are data included in an SGML document which are marked as being system-specific commands to the processing system. They do not return data to be included in the SGML document. This is what distinguishes PIs from SDATA. Processing instructions may be included inline in the document data or declared as entities.

prolog. The prolog of an SGML document is where the SGML DTD or document type declaration is found. All elements, attributes, entities, notations and other markup definitions are found in the DTD.

SDATA. SDATA is an SGML concept. It is data that may be included in an SGML document that is marked as system dependent. The SGML model assumes that SDATA content may be processed and that this processing will return data to be included in the document. This returned data cannot contain markup. This characteristic of returning data to be included in the document is what distinguishes SDATA form processing instructions. SDATA may only be defined in entity declarations.

SGML. See “Standard Generalized Markup Language”.

Standard Generalized Markup Language. SGML is an international standard for document markup. It is descended from IBM's GML markup language. Its key characteristics is the segregation of processing information for data and its precise definition of valid data organizations using a meta-markup language called a DTD. For more information see the references given in the front matter of this document.

Like GML, SGML has tags and attributes. In SGML, tags mark the start and end of elements (a concept that GML does not formally support). In addition, SGML supports the referencing of symbols and external data using the concept of entities.

symbol. In GML, a symbol is used to reuse, by reference, a string of data anywhere in the document. A DCF control word is used to define the symbol and then a symbol reference is used to indicate where in the document data the symbol's data should be placed.

tag. In SGML, a tag is used to mark the start or end of an element. In GML, a tag is a marker in the document data which is mapped to a set of DCF instructions, a macro.