SGML要素型定義の
部分的な交換・再利用・共有のための支援機構
(FPI[1])

[1]Facilities for Partial Interchanging, Reusing and Sharing of the SGML Element-type Definitions

目次

5.11 互換構文 ・・・

Chapter 0

序文

標準一般化マーク付け言語(SGML)は，文書の構造をマークにより記述し，電子文書データの交換を容易にする言語として規格化された。ある文書クラスの共通構造は，文書型定義(DTD)によって記述され，それに従ってマーク付けされた文書インスタンスが交換の対象となる。これまでに数多くのDTDが，さまざまな文書処理応用を目的として開発され，現在利用されている。この過程で次の問題点が明らかになってきた。

1. DTDの開発は労力のかかる作業である。類似したDTD，共通性のあるDTDが，個別に(互いに無関係に)作成されることにより，重複した無駄な労力が費やされている。
2. 質の高いDTDの開発は容易ではない。不用意に作成されたDTDは，関連する応用や利用者に，長い期間に渡り悪い影響を及ぼすことがある。
3. 異なるDTDに基づきマーク付けされた文書又は文書の一部分の交換は，一般に困難である。意味的，構造的には明らかに類似しているにもかかわらず，煩雑な変換などを経なければ，文書又は文書の一部分の交換又は共有ができない状況も多く見受けられる。
4. 同様な処理を行う文書処理応用が，DTDごとに作成される傾向がある。これもまた，重複した無駄な労力と低品質の応用による悪影響などの問題を引き起こす。
5. 周到に設計されたDTDであっても，運用の現場ごとに修正又は拡張が求められることが多い。修正又は拡張を行わずにDTDを使用し続けると，マークが誤用され，取り扱いが困難な文書データが生成・蓄積されることがある。

特に，官公庁など公共性が高く，大量の文書を扱う機関の文書データ電子化の局面において，これらの問題が顕在化している。公共性の高い文書は，その性質上，公開すべきものであり，生成容易性，交換性，再利用性が強く求められる。既に進められつつあるそれらの文書のSGML化に際しては，多数の互いに無関係なDTDが，交換性を阻害したり，応用開発の不要な費用を発生させないことへの注意が重要であり，そのための有効な方策が求められる。

このTRは，これらの問題点各項目において指摘した弊害・悪影響を軽減するために，SGML文書型定義の一部分、あるいはSGML文書型定義生成に利用する雛形を，交換，共有，再利用可能とするための概念及び手法を提供する。

Chapter 1

『JIS X 4151』の用語との相違

1.1 変更する訳語

以下の3つの訳語は、『JIS X 4151』ではなく『JIX X 4155』の訳語に従う。

原語	このTRでの訳語	『JIS X 4151』での訳語
token	トークン	字句
list	リスト	並び
instance	インスタンス	実現値

さらに、次の3つの訳語も変更する。

原語	このTRでの訳語	『JIS X 4151』での訳語
literal	リテラル	表記
parameter	パラメータ	引数
text	テキスト	文

1.2 変更する用語

以下の用語については、訳語の選択ではなく、用語そのものを変更する。

- 「共通識別子」(generic identifier)は、「要素型名」(element type name) とする。
- 属性の「宣言値」(declared value)は、属性値型(attribute value type)とする。

いずれも、より適切に意味を伝えるための変更である。

1.3 統合する用語、使わない用語

内容モデルを表現する形式を「モデル式」と呼ぶ。これは、「宣言内容」と「モデル群」を統合した概念・用語である。これに伴い、「宣言内容」という用語・構文変数は使わない。『JIS X 4151』の意味の「宣言内容」はモデル式の特別なものとみなす。例外(添加要素および排除要素)は使用しない。

1.4　区別する用語

『JIS X 4151』では、「属性定義リスト宣言」という用語・構文変数によって、要素に対する属性の宣言とデータ属性の宣言の両方を意味する。このTR では、「要素属性リスト宣言」、「データ属性リスト宣言」として区別する。また、単に「属性リスト宣言」と言った場合は、常に要素属性リスト宣言を意味する。

Chapter 2

単純要素型定義集合

単純要素型定義集合(S-ETDSと略記)は、要素宣言および(要素に対する)属性リスト宣言からなる集合である。S-ETDSはSGML構文に適合しており、まったく変更を加えずにDTDの一部となり得る。ただしS-ETDSでは、宣言の形式を一定にして交換性を高めるために、可能な限り構文を単純化する。

S-ETDSは要素集合(『JIS X 4151』の構文生成規則114)と類似しているが、記法宣言、データ属性リスト宣言、パラメータ実体参照、マーク区間宣言は含まれない。

□備考：

『JIS X 4151』の構文生成規則114は次のとおり。

[114] 要素集合=
(要素宣言
|属性リスト宣言
|記法宣言
|ds)*

属性リスト宣言にはデータ属性リスト宣言が含まれ、dsにパラメータ実体参照、マーク区間宣言が含まれる。

2.1 単純要素型定義集合の意義

S-ETDSは、??および??でそれぞれ定義するAS-ETDS、(一般の)ETDSの基礎となる。構文的には、S-ETDSはAS-ETDSの部分集合で、AS-ETDSはS-ETDSの部分集合になっている。ETDSを具体化すれば、AS-ETDSが得られる。さらに、単一化によりS-ETDSに変換が可能である。

一方、S-ETDSはSGML構文にも従っているので、このTRで定義する拡張された要素構造定義とSGMLのDTDとを関係付ける役割を持つ。

2.2 構文生成規則に関する注意

以下の各節で、S-ETDSを構文生成規則によって定義する。これらの構文生成規則は、『JIS X 4151』の構文生成規則を修正したものである。構文生成規則の番号は、対応する『JIS X 4151』の構文生成規則と同じ番号を使用する。ただし、このTRで修正した構文生成規則の番号には記号"S"が付してある。また、このTRで(修正ではなく)新たに追加した構文生成規則の番号には記号"S+"が付してある。

構文変数は、原則的に『JIS X 4151』と同じ名前を使用する。ただし、用語の変更に合わせて次の構文変数は変更する。

このTRの構文変数	『JIS X 4151』の構文変数
要素型名	共通識別子
トークン	字句
属性値型	宣言値
記法名群(属性値型として)	記法
モデルトークン	内容字句
モデル式	(なし)

構文生成規則中の構文変数の右側には、その構文変数を定義している構文生成規則の番号を示す。区切り子に関しては、右側に具体的な文字列を示す。

このTRでは、以下の構文生成規則を参照しており、これらは『JIS X 4151』、または附属書??に記述されている。

規則の番号	構文変数	『JIS X 4151』での構文変数
5	s	(同じ)
30	要素型名	共通識別子
33	属性値指定	(同じ)
50	SGML文字	(同じ)
55	名前	(同じ)
57	名前トークン	名前字句
91	注釈宣言	(同じ)
92	注釈	(同じ)
127	モデル群	(同じ)
130	要素トークン	要素字句
142	属性定義リスト	属性定義並び

2.3 共通の構成素

2.3.1 区切り子

S-ETDSにおいては、以下の規格参照区切り子を使用する。ただし、picaはこのTRで導入したものであり、規格参照区切り子ではない。

機能	文字列	説明	使用する所
and	&	接続子	モデル
com	--	注釈の開始と終了	注釈
cro	&#	文字参照の開始	属性値指定(リテラル)
grpc	)	群の終了	モデル, 群
grpo	(	群の開始	モデル, 群
lit	"	リテラルの開始と終了	属性値指定(リテラル)
lita	'	リテラルの開始と終了	属性値指定(リテラル)
mdc	>	宣言の終了	宣言
mdo	<!	宣言の開始	宣言
opt	?	出現標識	モデル
or	|	接続子	モデル, 群
pic	>	処理指令の終了	処理指令
pica	?>	処理指令の終了	処理指令
pio	<?	処理指令の開始	処理指令
plus	+	出現標識	モデル
refc	;	参照の終了	属性値指定
rep	*	出現標識	モデル
rni	#	予約名標識	モデル, 省略時値
seq	,	接続子	モデル

□備考：

pica(処理指令の終了の代替)はXMLとの互換性を確保するための例外である。XMLでの処理指令の終了は"?>"である。picとpicaは混在させて使用せず、どちらか一方だけを使用する。

□備考：

区切り子は、具象構文の区切り子集合(SYNTAX DELIM)で規定される。

2.3.2 予約名

S-ETDSにおいては、以下の規格参照予約名を使用する。

予約名	使用する所
ANY	モデル
ATTLIST	宣言の識別
CDATA	属性値型
ELEMENT	宣言の識別
EMPTY	モデル
ENTITIES	属性値型
ENTITY	属性値型
FIXED	属性省略時値
ID	属性値型
IDREF	属性値型

IDREFS	属性値型
IMPLIED	属性省略時値
NAME	属性値型
NAMES	属性値型
NMTOKEN	属性値型
NMTOKENS	属性値型
NOTATION	属性値型(記法名群)
NUMBER	属性値型
NUMBERS	属性値型
NUTOKEN	属性値型
NUTOKENS	属性値型
PCDATA	モデル
REQUIRED	属性省略時値

□備考：

予約名は、具象構文の予約名使用(SYNTAX NAMES)で規定される。

2.3.3 分離子

S-ETDSにおいては、Ee、パラメータ実体参照、マーク区間宣言を分離子として使わない。構文生成規則[65](psの定義)、[70](tsの定義)、[71](dsの定義)を変更する。

■構文生成規則1[S65] Ee、パラメータ実体参照はパラメータ分離子としない。

```
[S65]ps=
s ->[5]
|注釈 ->[92]
```

■構文生成規則2[S70] Ee、パラメータ実体参照はトークン分離子としない。tsはsと一致する。

```
[S70]ts=
s ->[5]
```

■構文生成規則3[S71] Ee、パラメータ実体参照、マーク区間宣言は宣言分離子としない。

```
[S71]ds=
s
|コメント宣言 ->[91]
|処理指令 ->[S44]
```

2.3.4 名前および名前トークン

S-ETDSにおいて、名前および名前トークンは、要素型名、属性名、属性値型の群(名前トークン群、記法名群)、省略時値としての名前トークン、省略時値としての一般実体名、省略時値の記法名として出現する。

□備考：

リテラル中に一般実体参照は使用しない。また、文字参照における機能名も使用しない。

名前文字として使用しない文字は次のとおり。

1. 規格参照区切り子を構成する区切り文字は名前文字とはしない。
2. 文字"¤"(単価記号)、文字"~"(チルド)は名前文字としない。

名前開始文字、名前文字に関して、このTRではこれ以外に何の制限も設けない。しかし、特定の応用や運用において、名前開始文字、名前文字を制限することを禁じるわけではない。

□備考：

SGML規格では、名前文字と区切り子の関係を次のように規定している。「名前文字が区切り子機能に割り当ててある場合、その名前文字は、既に名前トークンが始まっているのでなければ(名前文字と見ずに)その区切り子として認知する。既に名前トークンが始まっているのであれば、名前文字として扱う。」この規則を適用するのは混乱をまねく恐れがあるので、区切り文字と名前文字は重複しないこととする。

□備考：

文字"¤"と文字"~"は、(一般の)ETDSにおいて区切り文字として使用される。

名前および名前トークンの長さにも制限を設けない。しかし、特定の応用や運用において、名前および名前トークンの長さに制限を課すことを禁じるわけではない。(以下に登場する量的制限における「制限しない」、「制限を設けない」も同様に解釈する。)

□備考：

名前および名前トークンに使える文字は、具象構文の構文参照文字集合と命名方式(SYNTAX NAMING)で規定される。名前および名前トークンの長さ、それに関連する量は具象構文の量集合(SYNTAX QUANTITY)のNAMELENなどで規定される。

2.3.5 リテラル

S-ETDSにおいては、リテラルは属性の省略時値としてのみ使われる。リテラルには、データ文字と文字番号による文字参照を含んでよい。使用できるデータ文字には何の制限も設けない。リテラルの長さにも制限を設けない。

文字番号は、最初の128個までしか使えない。この128個は規格参照具象構文の構文参照文字集合(ISO 646-1983 IRV)で規定されるとする。文字番号による文字参照を許すのは、区切り文字が含まれる文字データの記述のためであり、文字参照の使用は、LIT(引用符、番号34)、LITA(アポストロフィー、番号39)、CROの最初の文字(アンパサンド、番号38)に限るのが望ましい。機能名による文字参照は使えない。

□備考：

リテラルに使用してよいデータ文字とその文字番号は、SGML宣言の文書文字集合で規定される。リテラルの長さ、それに関連する量は具象構文の量集合(SYNTAX QUANTITY)のLITLENなどで規定される。

リテラルの属性値としての解釈はSGML規格に従う。すなわち、それを囲むLITまたはLITAの組を取り除き、文字参照を置換し、RSを無視し、REおよびSEPCHARをSPACEに置換する。属性値型がCDATA以外のときは、先頭および末尾に現れるSPACEの連続をすべて無視し、その他の2個以上連続するSPACEを1個のSPACEに置換する。

2.3.6 群

名前トークン群、名前群で使用する接続子はOR接続子("|")に限る。

■構文生成規則4[S68] 接続子はOR接続子("|")に限る。

```
[S68] 名前トークン群=
_grpo, ->"("
ts*, ->[S70]
名前トークン, ->[57]
(ts*, ->[S70]
_or ->"|"
ts*, ->[S70]
名前トークン)*, ->[57]
ts*, ->[S70]
_grpc ->")"
```

■構文生成規則5[S69] 接続子はOR接続子("|")に限る。

```
[S69] 名前群=
grpo, ->"("
ts*, ->[S70]
名前, ->[55]
(ts*, ->[S70]
_or, ->"|"
ts*, ->[S70]
名前)*, ->[55]
ts*, ->[S70]
_grpc ->")"
```

群のなかのトークン(名前または名前トークン)の個数は制限しない。

□備考：

群のなかのトークンの個数は、具象構文の量集合(SYNTAX QUANTITY)のGRPCNTで規定される。

2.3.7 処理指令

■構文生成規則6[S44] 処理指令の終了にpica("?>")も使える。処理指令内で非SGML文字は使わない。

```
[S44] 処理指令=
_pio, ->"<?"
SGML文字*, ->[50]
(_pic ->">"
| _pico ->"?>")
```

picとpicaは混在させて使用せず、どちらか一方だけを使用する。処理指令の長さは制限しない。

□備考：

処理指令の長さは、具象構文の量集合(SYNTAX QUANTITY)のPILENで規定される。

2.4 正規化と比較

2.4.1 名前および名前トークンの正規化と比較

S-ETDSに出現する名前(予約名も含む)および名前トークンは同一性を判定できなくてはならない。名前トークン(名前を含む)の同一性の判定は、名前トークンの正規化に基づいて行う。正規化を行った後で、文字列として完全に一致すれば名前トークンは同一である。名前トークンの正規化は一意的に定まり、既に正規化された名前トークンを正規化しても変化しない操作でなくてはならない。

□備考：

文字の同一性は、その文字番号が等しいことである。2つの文字列の同一性は、先頭の文字から順に比較して、対応する全ての文字が等しく、長さも等しいことである。

このTRでは名前トークンの正規化の具体的な方法を規定しない。正規化の過程で、文字が無視されたり、一文字または複数の文字が別の文字に置換されたり、文字の順序が入れ替えられることがある。

2.4.2 文字データの正規化と比較

リテラルの内容である文字データは、SGML規格に従う解釈の後でさらに文字データの正規化を行う。このTRでは文字データの正規化の具体的な方法を規定しない。文字データの正規化は、名前トークンの正規化と同じ条件を満たす必要があるが、名前トークンの正規化とは異なる方法でもよい。

属性値の同一性は、それがCDATAであるときは、解釈された文字データの正規化後に文字列が完全に一致することであり、CDATA以外では、名前トークンの同一性を適用する。

2.5 単純要素型定義集合の構文

S-ETDSは，次の構文生成規則によって定義する。

■構文生成規則7[S+1]

```
[S+1] 単純要素型定義集合 =
(要素宣言 ->[S116]
|要素属性リスト宣言 ->[S141]
|ds)* ->[S71]
```

2.6 要素宣言の構文

2.6.1 宣言全体の構文

■構文生成規則8[S116]　タグ省略最小化は書かない。

[S116] 要素宣言=
_mdo, ->"<!"
"ELEMENT",
ps+, ->[S65]
要素型, ->[S117]
ps+, ->[S65]
内容モデル, ->[S126-1]
ps*, ->[S65]
_mdc ->">"

□備考：

タグ省略最小化を記述するかどうかは、SGML宣言の機構使用マーク最小化(FEATURES MINIMIZE)のOMMITTAGで規定される。

■構文生成規則9[S117]　要素型として指定できるのは単一の名前に限定する。名前群、付番要素、付番群は廃止。

[S117] 要素型=
要素型名 ->[30]

□備考：

付番を使用するかどうかは、SGML宣言の機構使用マーク最小化(FEATURES MINIMIZE)のRANKで規定される。

2.6.2　内容モデルの構文

内容モデルを表現する形式を"モデル式"とする。構文変数"宣言内容"を廃止して、宣言内容もモデル式の一種とする。CDATA, RCDATA、例外(添付要素および排除要素)は廃止。

モデル群に現れるトークンの総数、モデル群の入れ子の深さは制限しない。

□備考：

モデル群のなかのトークンの総数は、具象構文の量集合(SYNTAX QUANTITY)のGRPGTCNTで規定される。モデル群の入れ子の深さは、GRPLVELで規定される。

■構文生成規則10[S126-1]

[S126-1] 内容モデル=
モデル式 ->[S126-2]

■構文生成規則11[S126-2]

[S126-2] モデル式=
"EMPTY"
|"ANY"
|モデル群 ->[127]

□備考：

SGML規格では、内容モデルがあいまいであることを禁止している。このTRでは、あいまいな内容モデルを禁止しない。互換性の理由から、あいまいなモデルに対して警告をしてもよい。

■構文生成規則12[S129]　データタグ群は廃止。

```
[S129] 素モデルトークン=
(_rni, ->"#"
"PCDATA")
|要素トークン ->[130]
```

□備考：

データタグ群を記述できるかどうかは、SGML宣言の機構使用マーク最小化(FEATURES MINIMIZE)のDATATAGで規定される。

2.7　要素属性リスト宣言

■構文生成規則13[S141]　要素に対する属性定義に限定。

```
[S141] 要素属性リスト宣言=
_mdo, ->"<!"
"ATTLIST",
ps+, ->[S65]
結合要素型 ->[S72]
ps+, ->[S65]
属性定義リスト, ->[142]
ps*, ->[S65]
_mdc ->">"
```

属性定義リストが定義する属性の個数に制限は設けない。

□備考：

属性定義リストが定義する属性名の個数は、属性値型の名前トークンと共に、具象構文の最集合(SYNTAX QUANTITY)のATTCNTで規定される。

■構文生成規則14[S72]　名前群は廃止。

```
[S72] 結合要素型=
要素型名 ->[30]
```

■構文生成規則15[S147]　CURRENT、CONREFを廃止。

```
[S147] 省略時値=
((_rni, ->"#"
"FIXED",
ps+)?, ->[S65]
属性値指定) ->[33]
|(_rni, ->"#"
("REQUIRED"
|"IMPLIED"))
```

属性値指定として使われるトークンまたはリテラルの長さに制限は設けない。

□備考：

属性値指定が名前文字だけからなるリテラルのときは、LITまたはLITAで囲まなくてもよい。

2.8　単純要素型定義集合の構文要件

S-ETDSの形式的な構文は、上に構文生成規則で定義したとおりである。量的制約はない。S-ETDSは、あるDTDの一部とみなせなくてはならない。このため、構文生成規則で表現される以外のSGMLの構文要件も満たす必要がある。以下は、関係する構文要件の要約である。全ての構文要件の詳細はSGML規格を参照せよ。

- 群 — 同じトークンが、一つの名前トークン群(名前群)に2度以上現れてはならない。
- 要素宣言 — 同一の要素型(名)を持つ要素宣言が2個以上あってはならない。
- モデル群 — 一つのモデル群内の接続子は同一でなくてはならない。
- 属性リスト宣言 — 同一の結合要素型(名)を持つ属性リスト宣言が2個以上あってはならない。
- 属性リスト — 属性名は、一つの属性リストのなかで一度だけ指定することができる。
- 属性値型 — NOTATION属性値型は、その内容がEMPTYである要素型に対して指定してはならない。
- 属性値型 — 一つの結合要素型に対して、属性値型がIDまたはNOTATIONである属性はただ1個だけ指定できる。
- 省略時値 — 属性値型がIDのときの省略時値はIMPLIEDまたはREQUIREDでなくてはならない。
- 省略時値 — 属性値型が群の場合、省略時値はその群のトークンの一つでなくてはならない。
- 省略時値 — 省略時値として空の属性値表記("")を指定してよいのは、属性値型がCDATAのときだけである。

2.9 未定義の名前の扱い

S-ETDSでは、その宣言内に、次のような番号または名前が出現する可能性がある。

文字番号 省略時値のリテラル内に文字参照として出現する。

一般実体名 属性値型がENTITYまたはENTITIESである属性の省略時値として出現する。

記法名 属性値型としての記法名群のトークンとして出現する。また、記法属性の省略時値として出現する。

要素型名 内容モデル内に出現する。また、属性リスト宣言の結合要素型として出現する。

これらの番号、名前は次の場所で定義(導入)される。

文字番号	SGML宣言
一般実体名	(一般)実体宣言
記法名	記法宣言
要素型名	要素宣言

S-ETDSは、特定のSGML宣言を伴わず、一般実体宣言、記法宣言を含まない。このため、文字番号、一般実体名、記法名がS-ETDS内で定義されることはない。要素型名に関しても、要素宣言を持たない要素型名が出現する可能性がある。これら未定義の名前は次のように取り扱う。

1. 文字番号は最初の128個(0〜127まで)が使用可能であり、その意味は『ISO 646-1983』に従う。
2. 一般実体名、記法名は、あたかもそれが宣言されているかのように扱う。未定義エラーとはしない。
3. 内容モデル内または結合要素型として出現する要素型名は、対応する要素宣言が必ずしも存在しなくてもよい。(この条件は、SGMLにおいても許容されている。)

S-ETDSにおいては、宣言されてない実体名と記法名の使用は不正ではなく、その名前が未定の状態にある(名前と特定の実体・記法とが関係付けが、保留または遅延されている)と解釈する。

□**備考**：

一般実体宣言、記法宣言などを、S-ETDSに含めることはできないが、SGML宣言、実体集合(実体宣言の集合)、記法宣言の集合をS-ETDSに添えてもさしつかえない。それが望ましい場合もある。

□**例**：

```
    <!-- S-ETDSの例 -->
<!ELEMENT FOO (#PCDATA|BAR)* >
<!ATTLIST BAZ
    notation NOTATION (A|B|C) #IMPLIED
    figure ENTITY TheFigure >
<!-- 以上 -->
```

この例では、

- モデル内のBARの要素宣言がない
- 結合要素型BAZの要素宣言がない
- 記法名A, B, Cの記法宣言がない
- 一般実体TheFigureの実体宣言がない。

しかし、これは正しいS-ETDSである。

□備考：

未定義の名前を定義するために、要素宣言、一般実体宣言、記法宣言(データ属性リスト宣言を伴うこともある)を付け加えることができる。付け加える宣言により、名前の意味は変化する。このため、S-ETDSは、未定義の名前をプレイスホルダーとして持つともいえる。

2.10 SGML宣言との関係

S-ETDSは、インスタンスのマーク付けを直接に規定するものではない(その目的に使ってもさしつかえはないが)。一般的に有用な要素構造を記述し、交換、共有、再利用の対象とする目的を持つ。このため、S-ETDSは特定のSGML宣言に従う必要はない。しかし、このTRで定義したS-ETDSの構文に対して好都合なSGML宣言は考えられる。そのようなSGML宣言の一例を附属書?? (参考) に示す。

2.11 処理・運用上の規則および制限

S-ETDSを実際に使用する場合は、このTRで規定していない次の諸点を明確にする必要がある。

1. 処理指令 — 処理指令終了としてpicを使うかpicaを使うかを規定しなくてはならない。特定の応用に関連する範囲内では、どちらか一方に決めて使用しなくてはならない。
2. 名前 — 名前開始文字および名前文字を規定しなくてはならない。名前開始文字および名前文字は、できるだけ制限しないことが望ましい。
3. 名前 — 名前トークンの正規化の方法を規定しなくてはならない。
4. 文字データ — SGML文字(データ文字)を規定しなくてはならない。SGML文字は、できるだけ制限しないことが望ましい。(SGML文字は、注釈(注釈宣言も含む)、処理指令、省略時値のリテラルに使われる。)
5. 文字データ — 文字データ(文字列)の正規化の方法を規定しなくてはならない。(文字データの正規化は、省略時値として使われるリテラルの比較の際に使用される。)

S-ETDSを格納または処理する場合には、実行時に利用できる資源(メモリ、ディスクなど)の制約以外には制限を設けるべきではない。やむを得ぬ事情により制限を設ける場合でも、次の条件は満たさなくてはならない。

名前開始文字 英大文字(A〜Z)、英小文字(a〜z)は含まなくてはならない。

名前文字 英大文字(A〜Z)、英小文字(a〜z)、数字(0〜9)、"."(ピリオド)、"-"(ハイフン)は含まなくてはならない。

名前トークンの長さ 名前トークン(名前を含む)の正規化の後の長さ(文字数)として、40文字までの長さを許さなくてはならない。正規化された名前トークンの同一性を判定するときは、名前トークンを構成するすべての文字を考慮しなくてはならない(名前トークンを切り詰めてはならない)。

属性リストに含まれる属性定義の個数 属性値型の群に含まれるトークンの個数に関わらず、一つの属性リスト宣言に、50個までの属性定義を許さなくてはならない。(この値の意味は、具象構文の量集合(SYNTAX QUANTITY)のATTCNTとは異なる。)

宣言の総数 ひとつのS-ETDS内に、500個の要素宣言、500個の属性リスト宣言を許さなくてはならない。

処理指令の長さ 区切り子を除いて2000文字までの処理指令を許さなくてはならない。

その他 リテラルの長さ(240)、群に含まれるトークンの個数(32)、モデル群の入れ子の深さ(16)、モデル群に含まれるトークンの総数(96)は、規格参照量集合に従う(括弧内は規格参照量)。

以上の規則や制限の多く(すべてではない)は、SGML宣言を使って記述することができる。しかし、明確に規定ができるなら、SGML宣言以外の方法を使ってよい。

□**備考：**

S-ETDSを書く者は、この条件内に収まることを心がける必要はない。制限は、実行時の資源によるものだけにすべきであり、この条件は実用性の下限を示すに過ぎない。

□**備考：**

実行時の資源不足は回避しがたいものであり、この条件は実行時には適用しない。実行時の環境に応じて、実行時の制限は変動させるべきである。

2.12 応用DTDに組み入れる際の注意

S-ETDSは、そのまま全く変更を加えずに応用DTDの一部として利用できる場合もあるが、必要に応じて次の変形を施してもよい。

1. パラメータ実体 — S-ETDSでは、パラメータ実体宣言とパラメータ実体参照を一切用いない。応用DTDにおいてパラメータ実体参照を使用してもよい。このとき、DTDまたはその一部がS-ETDSに適合しているかどうかを判断するには、全てのパラメータ実体参照をパラメータ実体宣言に従って置換してから判断する。つまり、全てのパラメータ実体参照を置換する前処理を想定する。

2. マーク区間宣言 — S-ETDSでは、マーク区間宣言を一切用いない。応用DTDにおいては、状態見出し語がIGNOREまたはINCLUDEであるマーク区間宣言を用いてもよい。このとき、DTDまたはその一部がS-ETDSに適合しているかどうかを判断するには、IGNORE状態見出し語が指定されたマーク区間宣言を削除し、INCLUDE状態見出し語が指定されたマーク区間宣言をそのマーク区間(内容)で置換してから判断する。つまり、全てのマーク区間宣言を置換する前処理を想定する。

3. 区切り子および予約名 — S-ETDSにおいては、規格参照区切り子、規格参照予約名が用いられている。応用DTDがその他の区切り子集合、予約名集合を使っている場合は、区切り子、予約名の置換を行ってよい。

4. タグ省略最小化 — S-ETDSにおいては、要素宣言にタグ省略最小化を指定しない。応用DTDにおいては、タグ最小化を付加してもよい。

5. その他のタグ最小化、短縮参照 — S-ETDSは、インスタンスのマーク付けに関することは規定しない。必要に応じて、短縮タグの使用を許可したり、短縮参照対応表を付加してもよい。

□**備考：**

S-ETDSをそのままDTDの一部として使用する場合でも、要素宣言、属性リスト宣言、実体宣言、記法宣言、データ属性リスト宣言などを付け加える場合が多い。

Chapter 3

概単純要素型定義集合

概単純要素型定義集合(AS-ETDS)は、S-ETDSをわずかに拡張したものである。その拡張点は以下のとおりである。

1. モデル式として新しい予約名NULLを使用できる。
2. 属性値型として新しい予約名NULLを使用できる。
3. 同じ要素型(名)を持つ複数の要素宣言を許す。
4. 同じ結合要素型(名)を持つ複数の属性リスト宣言を許す。

□備考：

ASETDSは、後で定義する完全単一化により、FPI prohibit指令を持つSETDSに変換できる。

3.1 概単純要素型定義集合の構文

ASETDSの構文を、SGMLの構文またはSETDSの構文からの変更として定義する。

3.1.1 区切り子および予約名

区切り子の変更はない。予約名としてNULLが追加される。

3.1.2 要素宣言

モデル式の構文を変更。

■構文生成規則16[A 126-2] NULLを追加。

```
[A 126-2] モデル式=
"EMPTY"
|"ANY"
|"NULL"
|モデル群 ->[127]
```

3.1.3 属性リスト宣言

■構文生成規則17[A 147] 構文生成規則に変更はないが、次の要件を追加する。

属性値型がNULLの場合、省略時値としては、#IMPLIEDまたは#REQUIRED以外は指定できない。

```
[A 147] 省略時値=
((_rni, ->"#"
"FIXED",
ps+)?, ->[S65]
属性値指定) ->[33]
|(_rni, ->"#"
("REQUIRED"
|"IMPLIED"))
```

■構文生成規則18[A 145] NULLを追加。

```
[A 145] 属性値型=
"CDATA"
|"ENTITY"
|"ENTITIES"
|"ID"
|"IDREF"
|"IDREFS"
|"NAME"
|"NAMES"
|"NMTOKEN"
|"NMTOKENS"
|"NUMBER"
|"NUMBERS"
|"NUTOKEN"
|"NUTOKENS"
|"NULL"
|記法名群 ->[146]
|名前トークン群 ->[68]
```

3.2 新しい予約名NULLの意味

新しい予約名NULLは、要素宣言の内容モデル式として、また、属性リスト宣言の属性値型として使用できる。

□備考：

以下の説明において、インスタンス、パーザー、アプリケーションについての記述がある。これは、概念の説明のために言及しているだけで、そのような動作をするパーザー、アプリケーションの存在を仮定しないし、実装を要求もしない。

3.2.1 モデル式としてのNULL

モデル式にNULLが指定された場合は、そのモデルは、いかなるSGMLテキスト、あるいはデータによっても充足されない。このため、NULLを内容モデルとする要素型のインスタンスは存在し得ない。

例えば、fooがNULL内容モデルを持つ要素型なら、その要素宣言に従うSGML文書(または文書の一部)中では、foo要素のインスタンスの出現が禁止される。foo要素のインスタンスが出現すれば、その内容、出現した文脈にかかわらず誤りである。

□備考：

NULL内容と宣言された要素型(名)が、他の要素宣言の内容モデルに出現するのは誤りではない。内容モデルでの出現が禁止されるのではなく、インスタンスの出現が禁止される。

3.2.2 属性値型としてのNULL

属性値型NULLは、その属性の取り得る値が存在しないことを示す。属性値型がNULLと定義されているとき、対応する要素インスタンス(の開始タグ)にその属性指定が出現すれば誤りである。

属性値型NULLの省略時値が#REQUIREDである場合、その属性はいかなる値も取り得ないし、暗示的な(不定な)値も許されない。パーザーはその属性の存在をアプリケーションに伝えてはならない。

省略時値が#IMPLIEDである場合には、暗示的な(不定な)値のみが許される。このとき、パーザはアプリケーションにその属性の存在を伝えてもよい。ただし、不定値以外の値は取り得ない。

いずれの場合でも、要素インスタンスには属性指定が出現できないことにかわりはない。

3.3 概単純要素型定義集合の構文要件

AS-ETDSの構文生成規則で表現される以外のSGMLの構文要件は、SGMLの構文要件を次のように変更したものとする。以下の箇条に明確に示されてない事項についてはSGMLの構文要件に従う。

- 要素宣言 — 同一の要素型(名)を持つ要素宣言が2個以上あってもよい。
- 属性リスト宣言 — 同一の結合要素型(名)を持つ属性リスト宣言が2個以上あってもよい。
- 属性値型 — 同一の結合要素型(名)に対して属性リスト宣言が複数あっても、それらの属性リスト宣言のなかで属性値型がIDまたはNOTATIONである属性はただ1個だけ指定できる。
- 属性リスト — 属性名は、一つの属性リストのなかで一度だけ指定することができる。同一の結合要素型(名)に対して属性リスト宣言が複数あるとき、異なる属性リストにおいてなら、同じ属性名があってもよい。
- 属性値型 — NOTATION属性値型は、単一化した結果の内容がEMPTYである要素型に対して指定してはならない。
- 省略時値 — 属性値型がNULLのときの省略時値はIMPLIEDまたはREQUIREDでなくてはならない。

□備考：

同一の要素型(名)を持つ複数の要素宣言がなく、同一の結合要素型(名)を持つ複数の属性リスト宣言がなく、内容モデルと属性値型にNULLが出現しないAS-ETDSは、S-ETDSとなる。

□例：

```
  <!-- AS-ETDSの例 -->
<!ELEMENT foo (bar?, baz+)>
<!ELEMENT foo (bar, baz*)>
<!ELEMENT bar
a NULL #IMPLIED
>
<!ELEMENT zot NULL>
```

Chapter 4

同類宣言の単一化

4.1 (用語定義)

4.1.1 同類宣言(用語定義)

同じ要素型(名)を持つ要素宣言、同じ結合要素型(名)を持つ属性リスト宣言を同類宣言という。ある名前について、AS-SETDS内でその名前を要素型にもつ同類宣言を全て集めたものを同類宣言集合と呼ぶ。要素宣言と属性リスト宣言を区別する場合は、同類要素宣言集合、同類属性リスト宣言集合と呼ぶ。

従来のSGMLでは、同類要素宣言集合はたかだか1個の宣言からなる。AS-ETDSでは、同類宣言集合は任意個数の宣言からなる。

4.1.2 同類宣言の単一化(用語定義)

2個またはそれ以上の宣言からなる同類宣言集合を、同値な1個の宣言に変形することを(同類宣言の)単一化と呼ぶ。要素宣言の単一化と属性リスト宣言の単一化がある。同類宣言集合と1個の宣言が同値であるとは、宣言集合の全ての宣言を充足する要素インスタンスが、単一の宣言も充足し、その逆も成立することである。ただし、NULLモデル式およびNULL属性値型に関しては、要素および属性指定の明示的な禁止の意味を伴う。

4.1.3 同類宣言の完全単一化(用語定義)

AS-SETDSに対し、それ以上は単一化できなくなるまで単一化を行うこと。完全単一化を行うと、同じ要素型(名)を持つ要素宣言はたかだか1つとなり、同じ結合要素型(名)を持つ属性リスト宣言もたかだか1つとなる。

4.2 要素宣言の単一化

同類である2つの要素宣言に対して、その単一化を定義すれば十分である(単一化の操作は可換かつ結合的である)。

<!ELEMENT foo M>と<!ELEMENT foo N> (MとNはモデル式)が同類宣言だとして、モデル式Mとモデル式Nを正規表現と解釈して後、そのmeetである式をLとすると、<!ELEMENT foo L >が単一化を与える。Lの作り方は次の節で述べる。

□例：

```
  1番目の宣言    <!ELEMENT foo (bar?, baz+)>
2番目の宣言    <!ELEMENT foo (bar, baz*)>
               --------------------------
単一化した宣言 <!ELEMENT foo (bar, baz+)>
```

4.2.1 モデル式のmeet

MとNをモデル式として、M, Nに出現するすべての要素型名と、これらの要素型名とは異なる2つの名前c, xを準備する。ここで、c, xは具体的な名前ではなく、モデル式に出現するいかなる名前とも異なる2つの名前を一般的に表したものである。cは文字を意味し、xはモデル式には出現しない要素を総称的に意味する。以上のようにして構成した名前の集合をアルファベット(基本語彙)とする。アルファベットはN+2個の名前からなるとして、要素型名をa1, ..., aN とする。モデル式に、このアルファベット上の正規集合(の族)を対応付ける。

- 新しい予約語NULLには空なる正規集合を対応づける。
- 予約語EMPTYには空列だけからなる正規集合(これは空ではない)を対応づける。
- 予約語PCDATA(モデル式中では#PCDATA)には、正規表現c*に対応する正規集合{c}*を対応づける。
- 予約語ANYには、すべての列からなる集合を対応付ける。
- 要素型名には、その名前だけからなる単一集合である正規集合を対応づける。
- (M, N)には、Mに対応する正規集合とNに対応する正規集合の連接集合を対応づける。
- (M | M)には、Mに対応する正規集合とNに対応する正規集合の合併集合を対応づける。
- M* には、Mに対応する正規集合の列を任意回繰り返して得られる正規集合を対応づける。
- M? には、Mに対応する正規集合に空列を付け加えた正規集合を対応づける。

上の定義を適用する際に、必要があれば次の簡略化を行う。

- 3つ以上のトークンを接続子でつないだ群の形の式は、左から順にカッコでくくって2つのトークンからなる群の入れ子に変形する。例えば、(A, B, C)は((A, B), C)となる。
- M+は、(M, M*)に置き換える。
- (M & N)は、((M, N) | (M, N))に置き換える

以上の手順で、M, Nに対応する2つの正規集合が作られる。正規集合の集合としての共通部分が再び正規集合なので、その正規集合を生成する正規表現を構成し、対応するモデル式をLとする。Lは一意的に決まらないが、表現の違いだけで定義する内容はどのように構成しても変わらない。

正規集合または正規表現からモデル式を構成する際に、次の変形を行う。

- 正規集合が空である場合は、NULLとする。
- c*は#PCDATAに置き換える。
- どの要素型とも異なる名前xが出現する正規表現が得られた場合はANYに置き換える。

4.3 属性リスト宣言の単一化

2つの同類属性リスト宣言の場合に定義すれば十分である。

```
<!ATTLIST foo
        a1 ...
        ...
        aN ...>

<!ATTLIST foo
        b1 ...
        ...
        bM ...>
```

が同類宣言だとして、a1, ..., aN, b1, ..., bMに名前の重複がないなら、単に属性定義を寄せ集めて、一つの属性リスト宣言を構成する。次の形で単一化が得られる。

```
<!ATTLIST foo
        a1 ...
        ...
        aN ...
        b1 ...
        ...
        bM ...>
```

同じ属性名が出現したときは、その属性名に関する属性処方(属性値型と省略時値の組をそう呼ぶ)のmeetを作る。

□例：

```
    <!ATTLIST foo
          a IDREFS #IMPLIED
>
<!ATTLIST foo
          a NAME #REQUIRED
>
```

単一化

```
<!ATTLIST foo
          a IDREF #REQUIRED
>
```

4.3.1 属性処方のmeet

属性処方のmeetを求めるには、属性値型のmeetと省略時値のmeetを求める。省略時値のmeetは、属性値型のmeetによって修正されることもある。

属性値型の略記号を定める。

□備考：

略記号は、記述を簡単にするだけの目的で導入される。

CDATA	C
NMTOKEN	T
NMTOKENS	Ts
NUTOKEN	$
NUTOKENS	$s
NAME	N
NAMES	Ns
NUMBER	#
NUMBERS	#s
ENTITY	E
ENTITIES	Es
ID	I
IDREF	R
IDREFS	Rs
NULL	0
記法名群	n
名前トークン群	t

4.3.2 属性値型のmeet

属性値型のmeetは表によって与える。記述の都合上、1つの大きな表ではなく、5つの表に分割する。xとxのmeetはxであり、meetは可換であるため表の一部は省略してある(省略部分は - で示す)。

	C	T	Ts	$	$s	N	Ns	#	#s	0
C	-	-	-	-	-	-	-	-	-	-
T	T	-	-	-	-	-	-	-	-	-
Ts	Ts	T	-	-	-	-	-	-	-	-
$	$	$	$	-	-	-	-	-	-	-
$s	$s	$	$s	$	-	-	-	-	-	-
N	N	N	N	0	0	-	-	-	-	-
Ns	Ns	N	Ns	0	0	N	-	-	-	-
#	#	#	#	#	#	0	0	-	-	-
#s	#s	#	#s	#	#s	0	0	#	-	-
0	0	0	0	0	0	0	0	0	0	-

	C	T	Ts	$	$s	N	Ns	#	#s	0
I	I	I	I	0	0	I	I	0	0	0
R	R	R	R	0	0	R	R	0	0	0
Rs	Rs	R	Rs	0	0	R	Rs	0	0	0
E	E	E	E	0	0	E	E	0	0	0
Es	Es	E	Es	0	0	E	Es	0	0	0

	I	R	Rs	E	Es

```
I  -  -  -  -  -
R  O  -  -  -  -
Rs O  R  -  -  -
E  O  O  O  -  -
Es O  O  O  E  -

   C  T  Ts $  $s N  Ns #  #s O  I  R  Rs E  Es
n  n  n  n  O  O  n  n  O  O  O  O  O  O  O  O  O
t  t  t  t  O  O  t  t  O  O  O  O  O  O  O  O  O

  n  t
n -  -
t n  -
```

記法名群または名前トークン群のmeetは、群を有限集合と見なして、その共通部分をとる。共通部分が空となった場合は、NULLをmeetとする。

4.3.3 省略時値のmeet

省略時値のmeetは、基本的に下の表で与えられるが、属性値指定とFIXEDに関しては注意事項に従ってmeetを求める。

	IMPLIED	REQUIRED	属性値指定	FIXED
IMPLIED	IMPLIED	REQUIRED	属性値指定(1)	FIXED(1)
REQUIRED	REQUIRED	REQUIRED	REQUIRED	REQUIRED
属性値指定	属性値指定(1)	REQUIRED	属性値指定(2)	FIXED(2)
FIXED	FIXED(1)	REQUIRED	FIXED(2)	FIXED(3)

1. 省略時値として指定された属性値指定が、既に求められた属性値型のmeetに含まれない値であるときはREQUIREDとする。
2. 省略時値として指定された2つの属性値指定が一致しないときは、REQUIREDとする。
3. 省略時値として指定された2つの属性値指定が一致しないとき、一致しても既に求められた属性値型のmeetに含まれない値であるときはREQUIREDとする。

□備考：

省略時値として指定された2つの属性値指定が一致するかどうか(等しい値かどうか)を判定するには、文字データの正規化を用いる。

Chapter 5

要素種定義集合

5.1 用語定義

5.1.1 要素種

要素型から特定の要素型名を捨象したもの。内容モデルと属性リストの仕様を持つが、特定の要素型名は持っても持たなくてもよい。要素型は、要素型名が特定された要素種と考えられる。

5.1.2 テキスト種、テキストモデル、テキスト種名、テキスト種宣言、テキスト種参照

要素内容の一部または全部のパターン。内容と同様に、モデル式により記述される。テキスト種を記述するモデル式はテキストモデルであり、テキスト種宣言により、テキスト種の名前とモデルが結びつけられる。

テキスト種は、要素種宣言、テキスト種宣言のモデル式内で参照できる。参照の形式はテキスト種参照と呼ばれる。

5.1.3 種

要素種またはテキスト種。

5.1.4 不定群、不定値

不定な名前トークンまたは名前を含む群を表す形式を不定群と呼ぶ。不定なトークンは予約語#ETCで表す。不定の属性省略時値を表す形式を不定値と呼ぶ。不定値は予約語#VALUEで表す。

5.2 概要

要素種定義集合(ESDS)は、一般化された要素型である"種"を定義する宣言の集合であり、ESDSは、S-ETDS、AS-ETDSを拡張した構文を持つ。ESDSは次の特徴を持つ。

1. 要素型の一般化である要素種と、要素内容の一部(全部でもよい)のパターンを記述するテキスト種を導入する。ESDSは、要素種宣言、要素種に対する属性リスト宣言、テキスト種宣言の集まりである。
2. 要素種定義の手間を軽減するため、要素種の継承宣言を使用できる。
3. モデル式の構文が拡張されている。

4. 不定群と不定値が使用できる。

ESDSは、次の手順によりAS-ETDSに変換できる。

1. 不定群と不定値指定を具体化する。
2. 全ての要素種に対して、要素型名を割り当てる。
3. 継承宣言に従って宣言を追加する。(継承宣言は削除する。)
4. テキスト宣言に従ってモデル式を展開する。(テキスト宣言は削除する。)
5. 必要があれば、モデル式中の拡張構文要素を従来のモデル式に書き換える。
6. 要素種宣言、属性リスト宣言の単一化を行う。
7. 要素種宣言のモデル式内の要素種名を対応する要素型で置き換える。要素種宣言の要素種名を削除して要素宣言に書き換える。

5.3 新しい構文要素の解釈

ESDSは、構文的にA-SETDSと大きな違いはないが、その解釈は異なる。ESDSが定義するのは種であり、具体的な要素型名を規定しない場合がある。

□例：

```
<!ELEMENT [para] #ANY (#PCDATA|empha|@misc-inlines)*>
<!ELEMENT [empha] #ANY (#PCDATA)>
```

ESDSにおいては、要素種宣言の要素種、属性リスト宣言の結合要素種は要素型名ではなく要素種名である。モデル式に出現する名前も、要素種またはテキスト種を参照する。

構文上、要素種、テキスト種であることを明示する必要がある箇所では、特別な構文([name]と@name)を導入したが、それ以外の箇所では、通常の構文をそのまま使用する。

5.4 共通の構成素

5.4.1 区切り子

ESDSにおいては、4個の区切り子を追加する。

1. TSRI (text sort reference indicator) — モデル式内のテキスト種参照を識別する。
2. ESNO (element sort name open) — 要素種宣言において、要素種名を識別する。
3. ESNC (element sort name close) — 要素種宣言において、要素種名を識別する。
4. TIE (tie sgin) — 要素種宣言において、要素種名と要素型名が異なる名前になることを禁止する。

機能	文字列	説明	使用する所
tsri	@	テキスト種参照	モデル
esno	[	要素種名の開始	要素種宣言
esnc	]	要素種名の終了	要素種宣言
tie	-	関連強制記号	要素種宣言

5.4.2 予約名

AS-ETDSで使用した予約名に加えて、以下の予約名を使用する。一部の予約名の綴は既に使用されているものだが、ETDSにおいて新しい使用法が与えられる。

予約名	使用する所	説明
ANY	モデル	モデル式の内部でも使える
ANY	要素種	任意の要素型名を表す
ATTLIST	宣言の識別	要素種に対する宣言
ELEMENT	宣言の識別	要素種に対する宣言
EMPTY	モデル	モデル式の内部でも使える
ETC	群	不定の名前トークン群を表す
INHERIT	宣言の識別	継承宣言を開始する
NULL	要素種	要素型名が存在しないことを表す
NULL	モデル	モデル式の内部でも使える
TEXT	宣言の識別	テキスト種宣言を開始する
VALUE	省略時値	不定の属性値を表す

5.4.3 名前および名前トークン

ESDSにおいて、名前および名前トークンは、要素種名、テキスト種名、要素型名、属性名、属性値型の群(名前トークン群、記法名群)、省略時値としての名前トークン、省略時値としての一般実体名、省略時値の記法名として出現する。文字"@"(単価記号)、文字"~"(チルド)は区切り子文字なので、名前文字としない。

5.4.4 構文生成規則の番号

次の構文要素には、新しい構文規則番号を与える。

1. 要素種定義集合(ESDS) [G+1]
2. テキスト宣言 [G+2]
3. 継承宣言 [G+3]
4. 要素種名 [G+4]
5. テキスト種 [G+5]
6. テキスト種名 [G+6]
7. テキスト種参照 [G+7]
8. テキストモデル [G+8]

新しい構文要素でも、従来の構文要素と類似性が高いものは、対応する構文生成規則の番号を使用する。

5.4.5 群

群は不定群であってもよい。

□備考：

予約語ETCで表される不定トークン並びを含む群は不定群と呼ぶ。

■構文生成規則19[G68-1]

```
[G68-1]名前トークン群=
_grpo, ->"("
ts*, ->[S70]
名前トークン並び?
(ts*, ->[S70]
_or ->"|"
ts*, ->[S70]
(rni ->"#"
"ETC")?,
ts*, ->[S70]
_grpc ->")"
```

■構文生成規則20[G68-2]

```
[G68-2]名前トークン並び=
名前トークン, ->[57]
(ts*, ->[S70]
_or ->"|"
ts*, ->[S70]
名前トークン)*, ->[57]
```

□例：

```
(#ETC)
(foo|bar|#ETC)
```

■構文生成規則21[G69-1]

```
[G69-1]名前群=
_grpo, ->"("
ts*, ->[S70]
名前並び?
(ts*, ->[S70]
_or ->"|"
ts*, ->[S70]
(rni ->"#"
"ETC")?,
ts*, ->[S70]
_grpc ->")"
```

■構文生成規則22[G69-2]

```
[G69-2] 名前並び=
名前, ->[57]
(ts*, ->[S70]
_or ->"|"
ts*, ->[S70]
名前)*, ->[57]
```

5.5 要素種定義集合の構文

ESDSは，次の構文生成規則によって定義する。

■構文生成規則23[G+1]

```
[G+1] 要素種定義集合 =
(要素種宣言 ->[G116]
|要素種属性リスト宣言 ->[G141]
|テキスト種宣言 ->[G+2]
|継承宣言 ->[G+3]
|ds)* ->[S71]
```

5.6 要素種宣言の構文

5.6.1 宣言全体の構文

■構文生成規則24[G116]

```
[G116] 要素種宣言=
_mdo, ->"<!"
"ELEMENT",
ps+, ->[S65]
要素種, ->[G117]
ps+, ->[S65]
内容モデル, ->[G126-1]
ps*, ->[S65]
_mdc ->">"
```

■構文生成規則25[G117]　要素種として要素種名と要素型名を指定できる。要素型名は名前群であってもよい。

```
[S117] 要素種=
_esno, ->"["
要素種名, ->[G+4]
_esnc) ->"]"
ps*, ->[S65]
```

```
(要素型名 ->[30]
|名前群 ->[G69-1]
|(rni ->"#"
"ANY")
|(rni ->"#"
"NULL"))
```

名前群は不定群であってもよいが、要素種における不定群の使用は推奨しない。

□例：

```
  <!ELEMENT [empha] EM (#PCDATA)>
<!ELEMENT [empha] (EM|EMP) (#PCDATA)>
<!ELEMENT [empha] #ANY (#PCDATA)>
```

■構文生成規則26[G+4]

```
[G+4] 要素種名=
名前 ->[55]
```

5.6.2 内容モデルの構文

■構文生成規則27[G129]

```
[G129] 素モデルトークン=
(_rni, ->"#"
"PCDATA")
|(_rni, ->"#"
"EMPTY")
|(_rni, ->"#"
"ANY")
|(_rni, ->"#"
"NULL")
|種トークン ->[G130]
```

■構文生成規則28[G130]

```
[G130] 種トークン=
(要素種名 ->[G+4]
|テキスト種参照), ->[G+7]
出現標識? ->[132]
```

■構文生成規則29[G+7]

```
[G+7] テキスト種参照=
_tsri, -> "@"
テキスト種名 ->[G??]
```

□例：

```
(@header, @section*)
```

5.7　要素種属性リスト宣言

■構文生成規則30[G141]

```
[G141] 要素種属性リスト宣言=
_mdo, ->"<!"
"ATTLIST",
ps+, ->[S65]
結合要素種 ->[G72]
ps+, ->[S65]
属性定義リスト, ->[G142]
ps*, ->[S65]
_mdc ->">"
```

■構文生成規則31[G72]

```
[G72] 結合要素種=
要素種名 ->[G+4]
```

■構文生成規則32[G147]　不定値でもよい。

```
[G147] 省略時値=
((_rni, ->"#"
"FIXED",
ps+)?, ->[S65]
(属性値指定 ->[33]
|(_rni, ->"#"
"VALUE"))
|(_rni, ->"#"
("REQUIRED"
|"IMPLIED"))
```

□例：

```
<!ATTLIST doc
  dtd-version CDATA #FIXED #VALUE
>
```

5.8　テキスト種宣言

■構文生成規則33[G+2]

[G+2] テキスト種宣言=
_mdo, ->"<!"
"TEXT",
ps+, ->[S65]
テキスト種, ->[G+5]
ps+, ->[S65]
テキストモデル, ->[G+8]
ps*, ->[S65]
_mdc ->">"

□例：

```
<!TEXT heading (sec-num? main-h, sub-h?) >
```

■構文生成規則34[G+5]

[G+5] テキスト種=
テキスト種名 ->[G+6]

■構文生成規則35[G+6]

[G+6] テキスト種名=
名前 ->[55]

[G+8] テキストモデル=
モデル式 -> [A126-2]

5.9 継承宣言

■構文生成規則36[G+3]

[G+3] 継承宣言=
_mdo, ->"<!"
"INHERIT",
ps+, ->[S65]
(名前
|名前群), ->[S69]
ps+, ->[S65]
(名前
|名前群), ->[S69]
ps*, ->[S65]
_mdc ->">"

名前群は不定群であってはならない。

□例：

```
<!INHERIT (chapter|section) division>
```

5.10 要素型定義集合の構文要件

ESDSに関する構文要件は、AS-ETDSの場合と同様である。ただし、要素型は要素種に置き換えられる。

- 要素種宣言 — 同一の要素種(名)を持つ要素種宣言が2個以上あってもよい。
- 属性リスト宣言 — 同一の結合要素種(名)を持つ属性リスト宣言が2個以上あってもよい。
- 属性値型 — 同一の結合要素種(名)に対して属性リスト宣言が複数あっても、それらの属性リスト宣言のなかで属性値型がIDまたはNOTATIONである属性はただ1個だけ指定できる。
- 属性リスト — 属性名は、一つの属性リストのなかで一度だけ指定することができる。同一の結合要素種(名)に対して属性リスト宣言が複数あるとき、異なる属性リストにおいてなら、同じ属性名があってもよい。
- 属性値型 — NOTATION属性値型は、単一化した結果の内容がEMPTYである要素種に対して指定してはならない。
- 省略時値 — 属性値型が群であり、その群が不定でない場合、省略時値はその群のトークンの一つでなくてはならない。
- 省略時値 — 属性値型がNULLのときの省略時値はIMPLIEDまたはREQUIREDでなくてはならない。

5.11 互換構文

要素種宣言として、要素宣言も書けるようにするため互換構文を導入する。要素種宣言に互換要素種を使用してもよい。これにより、AS-ETDSをESDSだとみなす(埋め込む)ことができる。

■構文生成規則37[S117-2]

```
[S117-2] 互換要素種=
(_esno, ->"["
要素種名, ->[G+4]
_esnc) ->"]"
_tie ->""
ps*, ->[S65]
要素型名) ->[30]
|要素型名 ->[30]
```

互換要素種においては、要素種名と要素型名が同じ名前でなくてはならない。要素型nameだけの互換要素種は、[name] ~ nameと同じと解釈する。

□例：

```
  <!-- 以下の2つは同じ -->
<!ELEMEWT [EM] ~ EM (#PCDATA)>
<!ELEMEWT EM (#PCDATA)>
```

TIEは、構造定義としては意味を持たないが、要素種名と要素型名が常に一致しなくてはならないことを強制する。要素種名、要素型名のどちらか一方を改名して、異なる名前にしてはならない。

□備考：

TIEは、タグにおいて使用する共通識別子とモデル式内の構文的な範疇記号が同じであることを要求する。これは従来のSGMLの規則と一致する。TIEが指定されなければ、2つの名前はたまたま一致しただけで独立なものと解釈される。

Chapter 6

要素種定義集合の概単純要素型定義集合への変換

要素種定義集合は、一般的な構造を定義するのに好都合である。実際の応用に際しては、次の手順で概単純要素型定義集合に変換できる。

6.1 不定群、不定値の具体化

6.1.1 #ETCの具体化

不定群(#ETC)における#ETCは、1つ以上の名前または名前トークンからなる並びで置き換える。置き換えた結果は、確定した群となる。

(a|#ETC), (a|b|#ETC)などの形の不定群では、#ETCを空並びか、1個以上の名前または名前トークンからなる並びで置き換える。並びが空並びの時は、一番左側の"|"は取り去る。

□例：

以下の具体化は全て正しい。

```
(#ETC) --> (a)
(#ETC) --> (a|b)
(a|#ETC) --> (a|b)
(a|#ETC) --> (a)
```

6.1.2 #VALUEの具体化

#VALUEは、属性値指定により置き換える。

□例：

下の宣言は上の宣言の具体化である。

```
<!ATTLIST top-level
  dtd-version CDTA #FIXED #VALUE>

<!ATTLIST top-level
  dtd-version CDTA #FIXED '1.0 (1998 Feb)'>
```

6.2 要素型名の割り当て

要素種宣言に置いて、要素型が不定群を含む場合は不定群の具体化を行う。#ANY を含む場合は、任意の名前群で置き換える。

6.3 継承宣言に従って宣言を追加する

継承宣言

```
<!INHERIT (a1| ... |aN) (b1| ... |bM)>
```

は、N × M個の継承宣言をまとめて書いたものと解釈する。

```
<!INHERIT a1 b1>
<!INHERIT a1 b2>
...
<!INHERIT a1 bM>

<!INHERIT a2 b1>
<!INHERIT a2 b2>
...
<!INHERIT a2 bM>
...

<!INHERIT aN b1>
<!INHERIT aN b2>
...
<!INHERIT aN bM>
```

よって、次の単純な継承宣言に関して述べれば十分である。

```
<!INHERIT a b>
```

この継承宣言は、種名bに関する全ての要素種宣言と属性定義宣言は、種名aに関しても宣言されたものとみなすことを意味する。種名bに関する要素種宣言/属性定義宣言のおける種名をaに置き換え、要素型名は#ANYとした宣言を全て付け加える。

この作業は再帰的に行う。つまり、

```
<!INHERIT a b>
<!INHERIT b c>
```

であるとき、cに関する宣言もaに関する宣言として付け加わる。

□例：

```
  <!ELEMENT [c] NULL>
<!ATTLIST c
  id ID #IMPLIED
>
```

```
<!ELEMENT [b] bar (#PCDATA|d)*>
<!INHERIT a b>
<!INHERIT b c>
```

継承宣言に従って宣言を追加すると以下のようになる。

```
<!ELEMENT [c] NULL>
<!ATTLIST c
  id ID #IMPLIED
>
<!ELEMENT [b] bar (#PCDATA|d)*>
<!ATTLIST b
  id ID #IMPLIED
>
<!ELEMENT [a] #ANY (#PCDATA|d)*>
<!ATTLIST a
  id ID #IMPLIED
>
```

6.4 テキスト宣言に従ってモデル式を展開する

テキスト種宣言のモデル式がANY、EMPTY, NULLのときは、それぞれ(#ANY), (#EMPTY), (#NULL)に置き換える。これにより、モデル式は常に群とみなせる。テキスト種参照を群で置き換える。

□例：

3番目の宣言は、上の2つの宣言を展開したもの。

```
<!TEXT heading (sec-num? main-h, sub-h?) >
<!ELEMENT [section] #ANY (@heading?, para*)>

<!ELEMENT [section] #ANY ((sec-num? main-h, sub-h?)?, para*)>
```

6.5 モデル式中の拡張構文要素を従来のモデル式に書き換える

要素種宣言、テキスト種宣言には、#ANY, #EMPTY, #NULLが自由に出現する。これは次の手順で取り除くことができる。なお、結合子"&"と出現標識"+"は同値な式に置き換えて取り除いておく。3つ以上の項目を持つ群は2つの項目からなる群の結合として表現しておく。

6.5.1 #ANYの削除

#ANYは、出現可能な全ての要素種名と#PCDATAのOR結合モデル群に"?"出現標識を付けたモデル式で置換する。出現可能な要素種をa1, ..., aNとすれば、

```
(#PCDATA|a1|...|aN)*
```

である。出現可能な要素種名は状況により変わる。

6.5.2 #EMPTYの削除

モデル式の内側に出現する#EMTPYから順に次の置き換えを適用する。

1. (#EMPTY) ==>#EMPTY
2. #EMPTY* ==> #EMPTY
3. (A|#EMPTY) ==> A?
4. (#EMPTY|A) ==> A?
5. (A, #EMPTY) ==> A
6. (#EMPTY, A) ==> A

#EMPTYが消去されるか、または#EMPTYだけからなる式(#EMPTY)となる。モデル式自体が(#EMPTY)のときは、EMPTYに置き換える。

6.5.3 #NULLの削除

モデル式の内側に出現する#NULLから順に次の置き換えを適用する。

1. (#NULL) ==>#NULL
2. #NULL* ==> #NULL
3. (A|#NULL) ==> A
4. (#NULL|A) ==> A
5. (A, #NULL) ==> #NULL
6. (#NULL, A) ==> #NULL

#NULLが消去されるか、または#NULLだけからなる式(#NULL)となる。モデル式自体が(#NULL)のときは、NULLに置き換える。

口例：

#NULLを含むモデル式の簡約

```
((#NULL, a, b) | c)*
==> (#NULL | c)*
==> (c)*
```

6.6 要素種宣言の単一化

同類である2つの要素種宣言に対して、その単一化を定義すれば十分である(単一化の操作は可換かつ結合的である)。

<!ELEMENT [foo] A M>と<!ELEMENT foo B N>が同類宣言だする。A, Bは#ANY、 #NULL、名前または名前群、MとNはモデル式である。AとBを集合と解釈して、その共通部分を表す表現をCとする。モデル式Mとモデル式Nを正規表現と解釈して後、そのmeetである式をLとすると、<!ELEMENT [foo] C L >が単一化を与える。

	#ANY	#NULL	名前	名前群
#ANY	#ANY	#NULL	名前	名前群
#NULL	#NULL	#NULL	#NULL	#NULL
名前	名前	#NULL	名前,#NULL	名前,#NULL
名前群	名前群	#NULL	名前,#NULL	名前群, #NULL

6.7 属性リスト宣言の単一化

AS-ETDSの場合と同様である。

6.8 要素種の要素型への置き換え

要素種宣言の要素型が名前または名前群として具体化されたなら、内容モデル中に出現する要素種を、名前または名前群で置き換えてよい。ただし、要素型名が#NULLの場合は、臨時に要素型名を要素種名と同じとして、その要素型は出現できないものとする。

要素種宣言の要素種名を削除して要素宣言に書き換える。

Chapter 7

要素種定義集合の編成単位及び編成方式

7.1 編成単位の種類

複数の文書型，文書処理，又は文書体系において利用可能な一般的構造を記述した要素型/要素種定義集合は、それを次の**編成単位**にまとめて提供する。

- **モジュール**
- **セクション**
- **パッケージ**

モジュールの概念的な集まりとして**カテゴリ**を導入する。
それぞれの編成単位及びカテゴリの意味と使用目的は次のとおりとする。

モジュール 意味的又は構造的に関連する一揃(そろ)いの宣言をまとめたもの。モジュール内の宣言は，ある用途のために共に使われることを意図している。

セクション モジュールをさらに区分したもの。モジュールの規模が大きくなった場合は，それを複数のセクションに区分することによって，利用者の理解が容易になる。また，セクション単位で要素種定義集合を利用してもよい。

□備考：

特定の要素種定義集合に対する説明を記述する際，モジュールのセクション構成と説明文書の構成を一致させておくと参照に便利である。

パッケージ 保管や転送の目的で，モジュールをまとめたもの。パッケージを構成するモジュール間には，意味的又は構造的な関連が無くてもよい。

パッケージは，転送に際して一時的に作成してもよい。保管のための媒体の都合などで，パッケージを再編成又は変更してもよい。また，このTR外の方法でパッケージに相当する編成単位を生成し利用してもよい。

カテゴリ 互いに関連するモジュールの概念的な集まり。

□備考：

モジュールは，どのカテゴリにも属さなくてもよいし，複数のカテゴリに属してもよい。

□備考：

カテゴリは，それに属するモジュールを列挙することによって定義される集まりではない。むしろ，モジュールの分類又は概念的編成の方針に対して付けられた名前である。よって，カテゴリの意味及び目的などは，関係するモジュール提供者の協議と合意によって定めるのが望ましい。

これらの編成単位は，いずれも抽象的なものであり，物理的な媒体，格納方式，転送方式などを規定しない。

□備考：

「モジュール」，「セクション」，「パッケージ」，「カテゴリ」は，いずれも一般的によく使われる用語である。文脈によっては，混乱を招く恐れがある。このTRで規定された編成単位であることを明確に示すためには次の用語を使う。

- 要素種定義モジュール
- 要素種定義セクション
- 要素種定義モジュールパッケージ
- 要素種定義モジュールカテゴリ

7.2 編成単位の表現法式

抽象的な編成単位であるモジュール，セクション，パッケージを，具体的に表現するにはいくつかの方法がある。

ETDS文書型 このTRで定義するETDS文書型は，要素種定義集合の編成を表現する。ETDS文書型の文書インスタンスによって，編成単位を明示できる。

ETDS文書型に対応する処理指令規則 このTRで定義するETDS処理指令規則を用いると，ETDS文書型が規定するタグと同じ指定を，処理指令によって表現できる。

編集又は印刷の規則 要素種定義集合を，印刷物によって提供する場合は，編集又は印刷の規則によって，編成単位を示してもよい。この規則は，印刷物の適切な個所で，明確に記述するのが望ましい。

コメント規則 要素種定義集合内のコメント宣言を利用して編成単位を示してもよい。

□備考：

このTRで規定するのは，ETDS文書型及びETDS処理指令規則による編成単位の表現である。ETDS文書型の導入は，編成方式及び付与する属性を形式的に定義することが主な目的であり，ETDS処理指令の使用を推奨する。

編集又は印刷の規則，又はコメント規則を用いる場合には，提供者及び利用者又は応用の合意によってその規則を決める。

7.3 ETDS 文書型

7.3.1 公式公開識別子

??

ETDS DTDは，次の公式公開識別子によって参照する。

```
"+//...//DTD ETDS (Element type definition set)//JA"
```

モジュール，パッケージを文書要素種とするDTDの公式公開識別子は次のとおりとする。

```
"+//...//DTD
    ETDS-package (Element type definition set package)//JA"
"+//...//DTD
    ETDS-module (Element type definition set module)//JA"
```

7.3.2 文書型定義

EDTS DTDは，次のとおりとする。

```
<!-- ETDS DTD :
    要素種定義集合の編成単位及び編成方式
- -->
<!ELEMENT ETDS O O (ETDS-package|ETDS-module)* >

<!ELEMENT ETDS-package - - (ETDS-module|ETDS-info)* >
<!ATTLIST ETDS-package
    name         CDATA #IMPLIED
    provided-by CDATA #IMPLIED
    version      CDATA #IMPLIED
    date         CDATA #IMPLIED
    desc-string CDATA #IMPLIED
>

<!ELEMENT ETDS-module - -
    (#PCDATA|ETDS-section|ETDS-include|ETDS-require|ETDS-info)* >
<!ATTLIST ETDS-module
    name         CDATA #REQUIRED
    provided-by CDATA #IMPLIED
    version      CDATA #REQUIRED
    date         CDATA #IMPLIED
    desc-string CDATA #IMPLIED
    unique-id    CDATA #IMPLIED
    uid-naming-scheme   CDATA #IMPLIED
    category     CDATA #IMPLIED
>
```

```
<!ELEMENT ETDS-section - -
    (#PCDATA|ETDS-section|ETDS-include|ETDS-require|ETDS-info)* >
<!ATTLIST ETDS-section
    name        CDATA #IMPLIED
>

<!ELEMENT ETDS-include - O EMPTY>
<!ATTLIST ETDS-include
    ref-name    CDATA #IMPLIED
    ref-unique-id   CDATA #IMPLIED
    ref-uid-naming-scheme   CDATA #IMPLIED
    ref-locater CDATA #IMPLIED
    ref-loc-naming-scheme   CDATA #IMPLIED
    ref-description CDATA #IMPLIED
>

<!ELEMENT ETDS-require - O EMPTY>
<!ATTLIST ETDS-require
    ref-name    CDATA #IMPLIED
    ref-unique-id   CDATA #IMPLIED
    ref-uid-naming-scheme   CDATA #IMPLIED
    ref-locater CDATA #IMPLIED
    ref-loc-naming-scheme   CDATA #IMPLIED
    ref-description CDATA #IMPLIED
>

<!ELEMENT ETDS-info - O EMPTY>
<!ATTLIST ETDS-info
    name        CDATA #REQUIRED
    value       CDATA #IMPLIED
>

<!-- ETDS DTD 終わり -->
```

7.3.3 要素型の説明

ETDS ETDS文書型の文書要素型。パッケージとモジュールの任意の集まり。便宜上の要素である。

ETDS-package パッケージ。モジュールの任意の集まり。

ETDS-module モジュール。要素種定義集合を構成する宣言がその内容となる。いくつかのセクションに区分されてもよい。

ETDS-section セクション。要素種定義集合を構成する宣言がその内容となる。さらに下位のセクションに区分されてもよい。

ETDS-include 他のモジュール又はセクションの取り込み指令。この要素が出現した位置に，属性で指定されるモジュール又はセクションを取り込む。

ETDS-require 他のモジュール又はセクションの要求指令。属性で指定されるモジュール又はセクションを最終的には取り込む(出現位置で取り込む必要はない)。

ETDS-info 追加情報。パッケージ，モジュール，セクション内に任意に挿入できる記述，指定，又は指令など。その名前及び値の解釈及び使用法は，提供者又は応用が決める。

7.3.4 属性の説明

name パッケージ，モジュール，又はセクションの名前。SGMLで許される名前に限らず任意の文字列(CDATA属性値)を使用してよい。モジュール名の命名方式は，このTRでは規定しない。

追加情報(ETDS-info)のときは，情報を識別する名前。

provided-by パッケージ又はモジュールの提供者を示す。

version パッケージ又はモジュールのバージョンを示す。

date パッケージ又はモジュールの作成又は公開の日付を示す。

desc-string パッケージ又はモジュールの内容を短く記述した文字列。

unique-id モジュールを一意的に識別する(と期待できる)文字列。

uid-naming-scheme uniqu-idとして指定した名前が従っている命名方式を示す。『ISO 8879』(『JIS X 4151』)及び『ISO/IEC 9070』が規定するSGML公式公開識別子を使う場合は，値として"SGML-Formal-Public-Identifier"を指定する。

category モジュールの属するカテゴリを示す。複数のカテゴリ名を指定してもよい。カテゴリ名の命名方式，及び複数のカテゴリ名の併記方式は，このTRでは規定しない。

ref-name ETDS-requireにおいて，取り込むモジュール又はセクションを指定する名前又は番号または名前と番号の組み合わせ。

ref-unique-id ETDS-requireにおいて，取り込むモジュール又はセクションを指定する一意識別子。

ref-uid-nameming-scheme ETDS-include, ETDS-requireにおいて，ref-unique-idに指定した一意識別子が従っている命名方式。

ref-locater ETDS-include, ETDS-requireにおいて，取り込むモジュール又はセクションを指定する識別子。この識別子の解釈は，システム又は応用及び使用環境に依存する。ローカルシステムのファイル名，ネットワーク上の所在位置指定，モジュール又はセクションを生成するプログラム名などが用いられる。

ref-loc-naming-scheme ETDS-include, ETDS-requireにおいて，ref-locaterに指定した一意識別子が従っている命名方式。

ref-description ETDS-include, ETDS-requireにおいて，取り込むモジュール又はセクションを示唆又は説明する記述。他の属性が指定されていない場合は，利用者がこの記述を参考に，必要なモジュール又はセクションを入手で成する。

value 追加情報の値。

7.3.5 ETDS-package文書型及びETDS-module文書型

ETDS DTDからETDS要素宣言を取り除いたDTDによって定義される文書型をETDS-package文書型とする。

ETDS DTDからETDS要素宣言及びETDS-package要素宣言をを取り除いたDTDによって定義される文書型をETDS-module文書型とする。

□備考：

ETDS-package文書型及びETDS-module文書型を定義するDTDの公式公開識別子は，??で規定したとおりとする。

7.4 モジュール及びセクションの名前又は番号付け

モジュールには，それを参照するための名前(name属性値)を必ず付与しなくてはならない。一意識別子(unique-id属性値)も付与することが望ましい。一意識別子としては，可能な限り，他のモジュールの一意識別子と重複が起こらない文字列を選ぶ。

セクションには，必ずしも名前を付けなくてもよい。名前を付ける場合には，同一モジュール内でセクション名の重複が起こらない文字列を選ぶ。

名前のあるなしにかかわらず，セクションをモジュール内で一意に参照するため，次の規則によってセクション番号を定める。

1. モジュール内で，どのセクションにも属さない部分に番号0を与える。どのセクションにも属さない部分が，いくつかのセクションによって複数の部分に分断されていても同様とする。

   特に，モジュールがセクションを持たない場合は，モジュール全体に番号0を与える。

2. 最初に出現するセクションに番号1，その次のセクションに番号2を与える。以下同様の手続きで番号を与える。

3. 番号1のセクションに対し，これらの手続きを再帰的に適用して，1.0, 1.1, 1.2などの番号を与える。以下同様な手続きで，入れ子になったセクション全てに番号を与える。

セクション番号は，特定のセクションを参照する際に使用する。

7.5 処理指令による編成単位の表現

要素種定義集合の抽象的編成方式は，ETDS文書型定義で規定される。この文書型が規定するタグと1対1に対応する処理指令を使って編成単位を指定してよい。その規則は次のとおりとする。

1. 開始タグに対応する処理指令は次の形とする。

   <?ETDS *name attributes*>

   *name*は，対応する共通識別子から"ETDS-"を除いた残りの部分である。*attributes*は，タグ内の属性指定をそのまま使用する。

2. 終了タグに対応する処理指令は次の形とする。

   <?ETDS end-*name*>

*name*は，対応する共通識別子から"ETDS-"を除いた残りの部分である。

生成規則による定義は次のとおり。

[1] ETDS開始処理指令 =

pio, "ETDS", s+, 名前, 属性指定並び, s*, pic

[2] ETDS終了処理指令 =

pio, "ETDS", s+ "end-", 名前, s*, pic

□備考：

ETDS処理指令が正しいかどうかの判定は，対応するETDS文書インスタンスのマーク付けが正しいかどうかに帰着する。ただし，実際にETDS文書インスタンスに変換する必要はない。

□備考：

ETDS処理指令の解釈は，SGML構文解析の前に行なわれても後に行われてもよい。通常，処理指令は構文解析後にそのまま処理応用に渡されるが，ETDS処理指令においては，この原則に従わなくてもよい。

□例：

同じ編成構造が，タグと処理指令によってどのように表現されるかを示す。

タグを使用する場合:

```
<ETDS-packeage name=MyPackage>
  <ETDS-module name=ModuleA>
    <ETDS-require ref-name=ModuelX>
    <ETDS-section name=sect1>
    </ETDS-section>
    <ETDS-section name=sect2>
    </ETDS-section>
  </ETDS-module>
  <ETDS-module name=ModuleB>
  </ETDS-module>
</ETDS-package>
```

処理指令を使用する場合:

```
<?ETDS packeage name=MyPackage>
  <?ETDS module name=ModuleA>
    <?ETDS require ref-name=ModuleX>
    <?ETDS section name=sect1>
    <?ETDS end-section>
    <?ETDS section name=sect2>
    <?ETDS end-section>
  <?ETDS end-module>
  <?ETDS module name=ModuleB>
  <?ETDS end-module>
<?ETDS end-package>
```

Chapter 8

モジュールの組み込み処理

ETDS-include, ETDS-require要素によって定義される指令をinclude指令、require指令と呼ぶ。include指令とrequire指令はモジュールの組み込みの目的に使われるが、その使用法と意味は異なる。include指令は、その出現位置で提供側モジュールを展開し、利用者側モジュールの一部として埋め込んでしまう。提供者側モジュールの複製を挿入するのと全く同じである。それに対してrequire指令は、それまでの組み込みの状況に応じて処理するのが望ましい。

include指令/require指令は、編成された要素種定義集合を扱う利用者又は処理系が，それを解釈し，必要に応じて処理する。解釈及び処理の規則は以下のとおりとする。

8.1 includeの処理

□備考：

以下の手順は処理方法の説明のためのものであって、実際に処理系がこの通りの手順で処理を行なう必要性はない。処理結果が一致するならば、手順や方法をどのように実装しても構わない。

1. include指令が出現した位置で処理を行う。
2. 属性指定がまったくない場合は誤りとする。

   □備考：

   この誤りは，取り込みの処理が不可能なことを示す。マーク誤りではない。
3. 複数の属性指定がある場合は，次の優先順位に従う。
   (a) ref-locater属性の値が指定されていれば，その値からモジュールまたはセクションを特定する。そのとき，ref-loc-naming-schemeが指定されていれば，その値を利用する。
   (b) ref-unique-id属性の値が指定されていれば，その値からモジュールまたはセクションを特定する。そのとき，ref-uid-naming-schemeが指定されていれば，その値を利用する。
   (c) ref-name属性の値が指定されていれば，その値からモジュールまたはセクションを特定する。
   (d) ref-description属性の値が指定されていれば，その値からモジュールまたはセクションを特定する。

□備考：

ref-description の値から，モジュール又はセクションを特定することはできないことが多い。ref-description 属性は，主に利用者(人)がそれを読んで参考にすることを目的とする。

モジュールまたはセクションが特定できなければ，誤りとする。

□備考：

いったんモジュール又はセクションの特定ができれば，他の属性を解釈することは要求しない。しかし，他の属性から特定されるモジュール又はセクションが整合的(すべて同一又は同等なモジュール又はセクション)であるかを確認することは有用である。

4. モジュール又はセクションが特定できたなら，その内容である要素種定義集合を，include指令があった位置に取り込む。既に取り込まれたモジュール又はセクションは取り込みを行わないことが望ましい。
5. これらの処理は再帰的に行われる。すなわち，取り込んだモジュール又はセクションにETDS-include要素が出現すれば，それも処理の対象とする。

□備考：

再帰的な処理の正確な定義は次のとおり。

- モジュールにinclude指令が1つもない時(require指令はあってもよい)、include処理として行うべき事は何もない。
- モジュール内にinclude指令が出現する場合、全てのinclude連鎖の最大の長さをnとする。モジュールの最大include連鎖の長さがnだとすれば、その利用側モジュールの連鎖の長さはn-1である。n-1に関する手順は既に定義されて、利用者側モジュールに処理を行い、include独立モジュールが生成されていると仮定する。この仮定のもとで、必要なことは一段階の処理だけであるが、それは既に定義されている。

8.2 include無限連鎖の取り扱い

モジュール中にinclude指令が出現した場合には、対象モジュールがその位置に取り込まれる。取り込んだモジュールに別のinclude指令が含まれることがある。include指令で結ばれたモジュールの列をinclude連鎖とする。

include連鎖は無限に伸びることがあり得る。それは循環(ループ)が生じた場合である。循環が生じた場合には、全く同じモジュールのincludeが繰り返される。このとき、同一のモジュールが繰り返しincludeされる直前で連鎖を打ち切っても意味的な違いはない。無限連鎖が含まれる場合、意味的に同等で有限連鎖だけが含まれるinclude指令の木構造が構成可能な場合には、それを実際に構成し、無限連鎖を避けることが望ましい。ただし、この処理が困難な場合には次の処理を行なっても良い。

1. include無限連鎖を検出した段階でエラーとする。
2. include連鎖の長さに最大値を設け、その最大と越えた場合にエラーとする。

8.3 requireの処理

include指令が存在すると、利用者モジュールと提供者モジュールはinclude連鎖の関係におかれる。連鎖は、全体として木構造をなす。この連鎖の最初(木構造の値)の利用者モジュールを最上位モジュールという。これは、処理系の実行時の概念であり、モジュール自体に最上位かどうかの区別があるわけではない。処理系は最初に最上位利用者モジュールから処理を開始する。最上位モジュールとその他のモジュールはrequireの処理において異なった扱いを受ける。require指令は通常最上位モジュールにおいて処理される。

最上位モジュールにおけるrequire指令はそれを全てinclude指令に置き換える。続いてinclude処理を行なう。その結果、再び、require指令が出現すれば同様の処理を繰り返す。

□備考：

この方法では、最初のinclude木構造の各モジュールに含まれるrequireは全て最上位モジュールに集められて処理される。

require処理の繰り返しは無限に続くことも考えられるが、既に処理済みのものを記録することに無限の繰り返しを避けることができる。

8.4 重複取り込みの認識と処理

処理系はモジュール/セクション単位又は宣言単位で、一旦includeした宣言を繰り返しincludeしないような処置を行なのが望ましい。理論的には、無限のinclude連鎖又は無限のrequire処理/include処理手順が発生しなければ、重複したincludeは特に問題ではない。しかし、効率上好ましくないので、無限連鎖又は、無限繰り返しの検出には重複取り込みを監視するのは望ましい。

Appendix A

単純要素型定義集合に適したSGML宣言の例(参考)

SETDSに適した具象構文と、その具象構文を使用したSGML宣言の例をあげる。この具象構文およびSGML宣言は、あくまでも事例に過ぎない。

A.1 具象構文の例

以下の具象構文は、核具象構文をもとにしている。SETDSに適するものだが、インスタンスの処理はまったく考慮されていない。インスタンスの構文解析に使用する場合は変更する必要があるかもしれない。

注釈の意味は次のとおりとする。

- "for instance handling" — 実体やインスタンスの処理にのみ関係するので、特に指定しない。
- "not used" — 機構または構文を使用しないので指定する意味がない。
- "no need to change" — 変更する必要がない。

```
SYNTAX
 -- syntax-reference character set --
 SHUNCHAR
    CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
             16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
             127 255
 BASESET
    "ISO 646-1983//CHARSET
    International Reference Version (IRV) //ESC 2/5 4/0"
 DESCSET
    0   128 0
 -- function character identification --
 FUNCTION
    RE  13
```

```
    RS  10
    SPACE   32
    TAB SEPCHAR 9
-- naming rules --
NAMING
    LCNMSTRT ""
    UCNMSTRT ""
    LCNMCHAR "-."
    UCNMCHAR "-."
    NAMECASE
        GENERAL YES
        ENTITY  NO
-- delimiter set --
DELIM
    GENERAL SGMLREF
    SHORTREF NONE
-- reserved name use --
NAMES SGMLREF
-- quantity set --
QUANTITY SGMLREF
        ATTCNT 99999999
        -- ATTSPLEN for instance handling --
        -- BSEQLEN not used --
        -- DATAGLEN not used --
        -- DTEMPLEN not used --
        -- ENTLVL for instance handling --
        GRPCNT 99999999
        GRPGTCNT 99999999
        GRPLVL 99999999
        LITLEN 99999999
        NAMELEN 99999999
        -- NORMSEP no need to change --
        PILEN 99999999
        -- TAGLEN for instance handling --
        -- TAGLVL for instance handling --
-- END OF SYNTAX --
```

A.2 SGML宣言の例

以下のSGML宣言で参照している具象構文"-//INSTAC//SYNTAX Simple//EN"は、先のA.1で定義した具象構文とする。

「... 省略 ...」と書かれている部分には、データ文字として『JIS X 0208』の文字を使用できるように

するための文字集合記述(の文字集合区分)がはいる。
注釈の意味は次のとおりとする。

- "for instance handling" — 実体やインスタンスの処理にのみ関係するので、特に指定しない。
- "not used" — 機構または構文を使用しないので指定する意味がない。
- "do not change" — 変更してはならない。

```
<!SGML  "ISO 8879:1986"
-- document caharcter set --
CHARSET

BASESET
    "ISO 646-1983//CHARSET
    International Reference Version (IRV) //ESC 2/5 4/0"
DESCSET
    0   9   UNUSED
    9   2   9
    11  2   UNUSED
    13  1   13
    14  18  UNUSED
    32  95  32
    127 1   UNUSED
BASESET
    "ISO Registration Number 87//CHARSET
    JIS X 0208//ESC 2/6 4/0 ESC 2/4 2/9 4/2"
DESCSET
    ... 省略 ...

-- capacity set --
CAPACITY SGMLREF
    TOTALCAP 99999999
    -- ENTCAP not used --
    -- ENCHCAP not used --
    ELEMCAP 99999999
    GRPCAP 99999999
    -- EXGRPCAP not used --
    -- EXNMCAP not used --
    ATTCAP 99999999
    ATTCHCAP 99999999
    AVGRPCAP 99999999
    -- NOTCAP not used --
    -- NOTCHCAP not used --
```

```
    IDCAP 99999999
    IDREFCAP 99999999
    -- MAPCAP not used --
    -- LKSETCAP not used --
    -- LKNMCAP not sed --

-- concrete syntax scope --
SCOPE    DOCUMENT

-- concrete syntax --
SYNTAX PUBLIC "-//INSTAC//SYNTAX Simple//EN"

-- feature use --
FEATURES
    MINIMIZE
        DATATAG NO
        OMITTAG NO -- do not change --
        RANK    NO
        SHORTTAG YES
    LINK     SIMPLE  NO IMPLICIT NO EXPLICIT NO
    OTHER    CONCUR  NO SUBDOC   NO FORMAL   NO

-- application-specific information --
APPINFO NONE
>
```