Projekt FORAUS

Präsentationsregeln

Fassung vom 28-IIII-1994

4. Projektsemester
SS 1994

UNIVERSITÄT
BREMEN

Fachbereich Informatik

Projektbetreuung:
Prof. Dr. Bernd Krieg-Brückner, Michael Fröhlich, Mattias Werner

Inhaltsverzeichnis

1 Einführung

Bei der Semantik oder auch Bedeutungslehre handelt es sich um die Lehre von der inhaltlichen Bedeutung einer Sprache. In der Informatik bezieht man sich dabei fast ausschließlich auf Programmiersprachen. Die Präsentationsregeln des Projekts FORAUS, die das tatsächliche Aussehen eines logischen Elements einer DTD (Document Type Definition) auf einem Ausgabegerät bestimmen sollen, stellen ebenfalls eine Art Programmiersprache dar. Für eben diese soll auf den folgenden Seiten die Semantik beschrieben werden.

Das Dokument liegt hiermit in einer überarbeiteten und korrigierten fünften Fassung vor. Einzige wesentliche Neuerung ist das neue Schlüsselwort headheight, mit dem die Höhe einer Kopfzeilenbox festgelegt werden kann.

Noch ein paar Worte zur Typographie. Zur Verdeutlichung verschiedener Zusammenhänge werden in diesem Dokument verschiedene Schriftarten verwendet. Platzhalter einer Syntaxdefinition werden in *kursiver Schrift* gedruckt und in spitzen Klammern eingefaßt, z.B. $< Number >$. Beispiele oder Programmlisten wiederum werden in `Schreibmaschinenschrift` dargestellt. Ausgaben oder Ergebnisse, die aus der Anwendung der Präsentationsregeln resultieren, werden mit serifenloser Schrift gekennzeichnet.

Andreas Hinken
Michael Jäschke
Gunnar Kavemann
Rainer und Volker Schmidtke

2 Die Schlüsselworte

2.1 Die Syntax der Präsentationsregeln

Die formale Grundlage, auf die sich dieses Kapitel stützt, ist die kontextfreie Beschreibung der Grammatik der Präsentationsregeln mit Hilfe der Backus-Naur-Form. Besonders für die Gruppe PARSER ist das Wissen darüber essentiell. In der nachfolgenden Definition beschreibt `Number` eine dezimale Zahl, `Identifier` einen alphanumerischen Bezeichner. Die `AttrKeyWords`, `ValueKeyWords` sowie die `UnitKeyWords` und deren Semantik werden im Kapitel „Die Attribute“ genauer besprochen.

```
PR          = ViewBlock ObjectBlock { ObjectBlock }.
ViewBlock   = "VIEWS" "=" "(" ViewName {"," ViewName} ")" ";".
ObjectBlock = ( PhyObj | LogObj ) "{" [ObjDef] {ViewDef} "}" ";".
ObjDef      = TypeDef [ObjList] [AttrList].
TypeDef     = "TYPE" "{" TypeKeyWord "}"
ViewDef     = "VIEW" ViewName "{" ObjDef "}".
ViewName    = Identifier.
ObjList     = "OBJ" "{" [ ObjName {"," ObjName } ] "}".
AttrList    = "ATTR" "{" {Exp} "}".
ObjName     = PhyObj | LogObj { "&" LogObj } | "PAGEBREAK" | "PCDATA" | String.
PhyObj      = "!" Identifier.
LogObj      = "?" Identifier.
Exp         = AttrKeyWord "=" Value ";".
TypeKeyWord = Identifier.
AttrKeyWord = Identifier.
Value       = Number | Number Unit | ValueKeyWord {"," ValueKeyWord } | LogObj.
Unit        = UnitKeyWord.
UnitKeyWord = Identifier.
ValueKeyWord= Identifier.
```

2.2 Schlüsselworte und ihre semantische Bedeutung

Zentrale Aufgabe der Präsentationsregeln (PR) ist die Verknüpfung zwischen logischen Objekten und deren Erscheinungsbild in einer bestimmten Sicht. Hierzu werden die logischen Objekte mit physikalischen Auszeichnungen beschrieben und die Reihenfolge ihres Erscheinens festgelegt. Zusätzliche physikalische Objekte können zur Unterstützung der physikalischen Auszeichnung definiert werden.

VIEWS

Syntax: VIEWS = ($< ViewName >$ [, $< ViewName >, \ldots$]);

Verwendung: Definiert alle Sichten, die das System anzeigen können soll. Dieses Schlüsselwort muß am Anfang der PR stehen. Anwendung siehe VIEW.

Beispiel:

```
VIEWS = ( ASCII , WYSIWYG );
```

VIEW

Syntax: VIEW < *Sichtname* > { < *Objektbeschreibung* > }

Verwendung: Wird innerhalb einer Objektbeschreibung angewandt, um auszudrücken, daß nachfolgende Definitionen zu einer speziellen Sicht zugeordnet werden soll und diese nur in der/den angegebenen Sichten gelten. Stehen Objektdefinitionen ohne Einleitung mit dem Schlüsselwort VIEW in einem Objekt, gelten sie für alle Sichten. Es dürfen nur solche Sichtnamen hinter VIEW stehen, die zuvor in der VIEWS Definition eingeführt wurden.

Beispiel: In der WYSIWYG-Sicht wird das logische Objekt Dokument mit dem physikalischen Objekt Inhaltsverzeichnis sowie den logischen Objekten Titel und Kapitel dargestellt. In der ASCII-Sicht fehlt das Objekt Inhaltsverzeichnis.

```
VIEWS( WYSIWYG, ASCII );
                .
                .
?Dokument {
        VIEW WYSIWYG {
                TYPE { layout }
                OBJ { !Inhaltsverzeichnis, ?Titel, ?Kapitel }
                     }
        VIEW ASCII {
                TYPE { layout }
                OBJ { ?Titel, ?Kapitel }
                   }
          };
```

2.3 Logische Objekte

Es gibt logische und physikalische Objekte. Die logischen Objekte sind von der SGML-DTD abhängig. Zu jedem logischen Objekt der DTD kann eine entsprechende Definition in der PR erfolgen, um die Darstellung des Objektes im Dokument zu beschreiben. Die nicht in der PR aufgeführten Objekte werden bei der Darstellung nicht berücksichtigt. Über den Namen wird auf ein bestimmtes logisches Objekt der SGML-DTD verwiesen. Logische Objekte werden in den Präsentationsregeln durch ein vorangestelltes Fragezeichen gekennzeichnet. Wichtig: es dürfen in den Präsentationsregeln nur logische Objekte aus der DTD auch als solche gekennzeichnet werden.

Der Textinhalt eines Objektes wird durch Abarbeiten des SGML-Dokuments durch den Formatierer bzw. Interne Struktur aktualisiert. Ist ein Element der DTD mit dem Schlüsselwort PCDATA ausgestattet, dann kann in den Präsentationsregeln, ebenfalls durch das Schlüsselwort PCDATA, auf den dort hinterlegten Text (PCDATA = Parseable Character DATA) referenziert werden.

Ein logisches Objekt wird beschrieben durch seinen eindeutigen und nur einmal definierbaren Namen gefolgt von einer, in geschweiften Klammern gefassten, Objektdefinition. Diese besteht an erster Stelle immer aus der zwingenden Typdeklaration. Dieser kann ein Satz von logischen und physikalischen Folgeobjekten, eingeleitet durch das Schlüsselwort OBJ, und Attributen, eingeleitet durch ATTR, folgen. Folgeobjekte oder Attribute müssen nicht in jeder Objektdefinition angegeben werden. Sie können ganz wegfallen oder nur teilweise beschrieben sein.

Unter Verwendung des Schlüsselwortes VIEW lassen sich Sätze für unterschiedliche Sichten auf das Dokument bilden. Zu beachten ist dabei, daß die Attributdefinitionen nicht nur im Objekt selbst

Gültigkeit haben, sondern an die Folgeobjekte vererbt werden. Ist dies nicht erwünscht, so muß der Attributwert im Folgeobjekt neu gesetzt werden.

? (logisches Objekt)

Syntax: ?< *logischer_Objektname* >{ < *Objektdefinition* > };

Beispiel:

```
?Protokoll {
# Folgende Definitionen werden fuer alle Sichten definiert!
        TYPE { layout }

        OBJ { !Titel, ?TOPs }
# Das logische Objekt Protokoll besteht aus einem
# physikalischen Titel-Objekt gefolgt von
# logischen TOPs-Objekten.

        ATTR { fontsize=10pt; }
# Das logische Objekt Protokoll verwendet und vererbt die
# Schriftgroesse 10pt fuer alle Folgeobjekte.
          };
```

2.4 Physikalische Objekte

Physikalische Objekte werden in den PR dazu benutzt, um gleiche Attribut- /Objektdefinitionen in mehreren logischen Objekten zu verwenden. Wie schon bei logischen Objekten (s. oben) haben physikalische Objekte in den PR eindeutige Namen, werden jedoch statt mit einem Fragezeichen mit einem vorausgehenden Ausrufezeichen besonders gekennzeichnet. Die weitere Anwendung und die Eigenschaften entsprechen dem der logischen Objekte. Sie können aber, im Gegensatz zu den logischen Objekten, über PCDATA keinen Textinhalt abbilden.

Bemerkung: Physikalische Objekte haben für die Darstellung des Dokumentes keine Bedeutung, wenn sie nicht durch logische Objekte verwendet werden.

! (physikalisches Objekt)

Syntax: !< *Objektname* > { < *Objektdefinition* > };

Beispiel:

```
!Titel {
        TYPE { text }
        OBJ { "Protokoll des Plenums" , PAGEBREAK }
# Der Titel besteht aus einer Zeichenkette gefolgt
# von einem Seitenumbruch.

        ATTR { fontsize=16pt; }
# Fuer den Titel wird eine Zeichengroesse von 16 Punkt bestimmt.
       };
```

TYPE

Syntax: TYPE < *objecttype* >

Verwendung: Mit diesem Schlüsselwort wird für jedes Objekt festgelegt, von welchem Typ es ist. Eine Typisierung ist für jedes Objekt zwingend vorzunehmen! Diese entscheidet darüber, wie der Formatierer mit diesem Objekt umgeht. TYPE steht immer vor OBJ und ATTR Deklarationen. Gültige Objekttypen sind:

< ***objecttype*** >	**Bedeutung**
catalog	dient dem Erstellen von Verzeichnissen
counter	definiert einen Zähler; Zählobjekte finden Anwendung beispielsweise beim Durchnummerieren von Kapiteln
index	Objekte vom Typ Index bauen ein Indexverzeichnis auf
layout	Objekte, die lediglich zum Anordnen anderer Objekte dienen
note	Noten begegnen uns hauptsächlich in einer ganz bestimmten Form: der Fußnote
reference	Referenzobjekt
text	einfaches Textobjekt

Beispiel: [1] Im ersten Beispiel soll ein einfaches Textobjekt definiert werden. Es wird angenommen, daß in der DTD ein logisches Objekt Ueberschrift existiert:

```
?Ueberschrift{
      TYPE{ text }                        # einfaches Textobjekt
      OBJ{ PCDATA }                       # nimm Text aus logischem
                                          # Objekt Ueberschrift
      ATTR{
            fontsize = 14pt;              # Groesse der Zeichen
            fontstyle = bold;             # Fettschrift
          }
             };
```

[2] Im zweiten Beispiel soll ein Inhaltsverzeichnis definiert werden, welches aus Kapitelüberschriften besteht.

```
!Inhaltsverzeichnis{
      TYPE{ catalog }
      ATTR{
            linkobj = ?Ueberschrift;    # s. Beispiel [1]
            fontsize = 12pt;            # ersetzt 14pt aus [1]
          }
                   };
```

$\boxed{3}$ In diesem Beispiel wird ein Zähler für Kapitelüberschriften erzeugt. Jedesmal, wenn das Objekt verwendet wird, wird es um eins hochgezählt.

```
!ZaehlerUeberschrift{
      TYPE{ counter }                  # def. Zaehlobjekt
      ATTR{
            numstartvalue = 0;         # Beginne mit 0
            numincrement = ?Kapitel;   # Inkrementiere immer beim
                                       # Auftreten des log. Objektes
            numsymbol = arabic;        # Ausgabe in
                                       # arabischen Ziffern
          }
                    };
```

$\boxed{4}$ Der Objekttyp index ist dem Verzeichnis (catalog) nicht ganz unähnlich. Ein Indexverzeichnis könnte wie folgt deklariert werden:

```
!Index{
      TYPE{ index }
      ATTR{
            linkobj = ?Indexeintrag;   # ein logisches Element
            fontsize = 10pt;           # Groesse Zeichensatz 10pt
          }
      };
```

$\boxed{5}$ Nachstehendes Beispiel soll den Objekttypen note anhand von Fußnoten erläutern. Das logische Objekt Fussnote wird mit Attributen versehen, die dafür sorgen, daß alle Noten auf der Seite ihres Erscheinen angezeigt werden.

```
?Fussnote{
      TYPE{ note }
      ATTR{                            # Fussnote ist vom Typ note
            keepwith = page;           # Note auf Seite ihres
                                       # Erscheinens
            notesymbol = arabic;       # arab. Zahlendarstellung
            notesymbolstart = ?Kapitel; # Nummerierung beginnt
                                       # mit jedem logischen
                                       # Objekt Kapitel neu
            fontsize = 8pt;            # Zeichensatzgroesse 8pt
          }
       };
```

ATTR

Syntax: ATTR { < *SatzVonAttributen* > }

Verwendung: Zu jedem Objekt (und Sichten) kann ein Satz von Attributen definiert werden. Das Schlüsselwort ATTR leitet diese Sequenz ein. Der Block von Attributzuweisungen wird durch geschweifte Klammern eingeschlossen.
Beachte: Hinter jeder Attributzuweisung steht eine Semikolon als Abschluß!

Beispiel:

```
ATTR {
       font=Times;
       fontsize=10pt;
     }
```

(Kommentare)

Syntax: [< *Kommando* >] # < *Kommentar* >

Verwendung: Kommentare können alleinstehend in einer Zeile oder am Ende einer Kommandozeile stehen und werden nicht ausgewertet. Ein Kommentar wird durch das # Symbol eingeleitet und endet mit dem Zeilenumbruch, so daß die nachfolgende Zeile — bei längeren Anmerkungen — mit einem weiteren Ziffernzeichen eingeleitet werden muß.

Beispiel:

```
fontsize=10pt; # Dieses ist ein Kommentar
# In dieser Zeile steht der Kommentar am Anfang und
# wurde in dieser Zeile fortgesetzt.
```

OBJ

Syntax: OBJ { [< *Objektname* >] [, < *Objektname* >] ...] }

Verwendung: Das Schlüsselwort OBJ wird in einer Objektdefinition zum Festlegen der Folgeobjekte benutzt. Die Reihenfolge der Nennung in der OBJ-Definition bestimmt die Reihenfolge der Auswertung durch den Formatierer (also das Auftreten im Dokument). Wird kein Folgeobjekt angegeben, so ist das aktuelle Objekt das Endobjekt in der Objekthierarchie. Mögliche Folgeobjekte sind:

physikalische Objekte — eingeleitet durch Ausrufezeichen (!)
logische Objekte — eingeleitet durch Fragezeichen (?)
PCDATA -Schlüsselwort (nur bei logischen Objekten möglich)
PAGEBREAK -Schlüsselwort (erzwungener Seitenumbruch)
Zeichenketten mit einleitenden und ausleitenden Anführungsstrichen (")
zusammengesetzte Ausdrücke aus logischen Objekten mit alternativem Auftreten der Objekte (&-Operator)

Beispiel: $\boxed{1}$ Das Objekt Kapitel hat die Folgeobjekte Kapitelzaehler (physikalisches Objekt), Ueberschrift (logisches Objekt) und den eigentlichen Textinhalt zum Kapitel-Objekt, gefolgt von entweder einem logischen Objekt Unterkapitel oder einem logischen Objekt Liste. Die Darstellungsreihenfolge von den Objekten Unterkapitel und Liste entspricht der Reihenfolge des Auftretens im SGML-Text.

```
?Kapitel {
        TYPE { text }
        OBJ { !Kapitelzaehler, ?Ueberschrift, PCDATA, ?Unterkapitel & ?Liste }
         };
```

$\boxed{2}$ Das „Minimal-“ Inhaltsverzeichnis besteht in diesem Beispiel aus dem Text “Inhaltsverzeichnis“ gefolgt von einem Seitenumbruch.

```
!Inhaltverzeichnis {
        TYPE { catalog }
        OBJ { "Inhaltsverzeichnis", PAGEBREAK }
                   };
```

PCDATA

Syntax: PCDATA

Verwendung: Durch das Schlüsselwort PCDATA kann in einer Objektdefinition das Auswertung des Textinhalts des logischen Objekts beeinflußt werden. Hierdurch ist es möglich, Folgeobjekte vor dem Textinhalt des aktuellen Objektes zu setzen. Falls dieses Schlüsselwort nicht angegeben wird, ist der Textinhalt des aktuellen Objekts auf jeden Fall vorangig vor den Folgeobjekten.

Beispiel: $\boxed{1}$ Beispiel ohne PCDATA: Der Text zum logischen Objekt Kapitel wird zuerst dargestellt, dann erst der Text des/der Objekts/Objekte Unterkapitel.

```
?Kapitel {
        TYPE { text }
        OBJ { ?Unterkapitel }
         };
```

$\boxed{2}$ Beispiel mit PCDATA: Der Text zum logischen Objekt Kapitel steht nach dem Objekt Unterkapitel.

```
?Kapitel {
        TYPE { text }
        OBJ { ?Unterkapitel, PCDATA }
         };
```

PAGEBREAK

Syntax: PAGEBREAK

Verwendung: Durch das Schlüsselwort PAGEBREAK kann in einer Objektdefinition der Seitenumbruch hinter dem vorangehenden physikalischen oder logischen Objekt erzwungen werden.

Beispiel: Im Objekt Dokument erscheint zunächst das physikalische Objekt Inhaltverzeichnis. Das darauf folgende Objekt Haupttext erscheint auf der nächsten Seite.

```
?Dokument {
        TYPE { catalog }
        OBJ { !Inhaltsverzeichnis, PAGEBREAK, ?Haupttext }
          };
```

3 Die Attribute

3.1 Dokument

keepwith

Syntax: keepwith = < *object* > | page | nil;

Verwendung: Hiermit können zwei Objekte der Präsentationsregeln so miteinander verbunden werden, daß sie beim Setzen durch den Formatierer nicht auseinandergerissen werden. < *object* > steht für das logische oder physikalische Objekt, mit dem das Objekt, für das keepwith definiert wird, verbunden wird. Stattdessen kann aber auch page angegeben werden, um dafür zu sorgen, daß das Objekt unbedingt auf der aktuellen Seite plaziert wird (z.B. Fußnoten). Bei nil wird das keepwith Attribut nicht weiter beachtet.

Beispiel: Es wird ein Kapitel, bestehend aus den logischen Elementen Ueberschrift und Absatz, definiert. Mit dem Attribut keepwith kann nun verhindert werden, daß die Überschrift auf Seite N und der zugehörige Absatz auf Seite N+1 steht:

```
?Kapitel{
          TYPE{ layout }
          OBJ{ ?Ueberschrift, ?Absatz } # Reihenfolge festlegen
        };

?Ueberschrift{
          TYPE{ text }                  # einfaches Textobjekt
          OBJ{ PCDATA }
          ATTR{
                fontsize = 14pt;        # Zeichensatzgroesse 14pt
                fontstyle = bold;       # Fettschrift
              }
             };

?Absatz{
        TYPE{ text }                    # einfaches Textobjekt
        OBJ{ PCDATA }
        ATTR{
              fontsize = 12pt;          # Zeichensatzgroesse 12pt
              fontstyle = plain;        # kein besonderer Schriftschnitt
              keepwith = ?Ueberschrift; # Ueberschrift und
                                        # Absatz nicht trennen
            }
       };
```

position

Syntax: position = < *objpos* >;

Verwendung: Mit position wird ein Objekt grob auf der Seite positioniert. < *objpos* > kann folgende Werte annehmen:

< ***objpos*** >	**Positioniert Objekt**
leftline	am linken Blattrand der Seite
leftpage	auf der „linken“ Seite
rightline	am rechten Blattrand der Seite
rightpage	auf der „rechten“ Seite
topofcolumn	an den Anfang einer Spalte
topofpage	an den Anfang einer Seite
nextline	an den Anfang der nächsten Zeile
head	in der Kopfbox der Seite
body	in der Hauptbox der Seite
foot	in der Fußbox der Seite
lastpos	an der letzten aktuellen Stelle

Beispiel: Das einfache Beispiel sorgt dafür, daß ein Absatz immer am Anfang einer Zeile begonnen wird.

```
?Absatz{
      TYPE{ text }
      OBJ{ PCDATA }
      ATTR{ position = nextline; }
       };
```

3.2 Seitenlayout

Allen Schlüsselwörtern für das Seitenlayout, die in der ATTR-Umgebung definiert werden, folgt eine $<\mathit{number}>$ gefolgt von einem $<\mathit{unitkeyword}>$. Als $<\mathit{unitkeyword}>$ stehen zu Verfügung:

$<\boldsymbol{unitkeyword}>$	**Bedeutung**
mm	Millimeter
cm	Zentimeter
in	Zoll
pt	Point
em	zum jeweiligen Zeichensatz relatives Maß für die Breite eines M
ex	zum jeweiligen Zeichensatz relatives Maß für die Höhe eines X

evensidemargin

Syntax: evensidemargin = $<\mathit{number}>$ $<\mathit{unitkeyword}>$

Verwendung: Setzt den linken Abstand aller drei Boxen auf ungeraden Seiten.

Beispiel: `evensidemargin = 2.5cm`

oddsidemargin

Syntax: oddsidemargin = $<\mathit{number}>$ $<\mathit{unitkeyword}>$

Verwendung: Setzt den linken Abstand aller drei Boxen auf geraden Seiten.

Beispiel: `oddsidemargin = 1in`

footheight

Syntax: footheight = $<\mathit{number}>$ $<\mathit{unitkeyword}>$

Verwendung: Setzt die Höhe der Fußzeilenbox

Beispiel: `footheight = 3cm`

footsep

Syntax: footsep = $<\mathit{number}>$ $<\mathit{unitkeyword}>$

Verwendung: setzt den Abstand der Unterkante der Rumpfbox und der Oberkante der Fußzeilenbox.

Beispiel: `footsep = 1.5 ex`

headheight

Syntax: headheight = $<\mathit{number}>$ $<\mathit{unitkeyword}>$

Verwendung: Bestimmt die Höhe der Kopfzeilenbox.

Beispiel: `headheight = 2cm`

headmargin

Syntax: headmargin = < *number* > < *unitkeyword* >

Verwendung: Setz den linken Rand der Kopfzeile relativ zur odd- oder evensidemargin.

Beispiel: `headmargin = -1cm`

headsep

Syntax: headsep = < *number* > < *unitkeyword* >

Verwendung: Setzt den Abstand zwischen der Unterkante der Kopfzeilenbox und der Oberkante der Rumpfbox.

Beispiel: `headsep = 3.5 ex`

topmargin

Syntax: topmargin = < *number* > < *unitkeyword* >

Verwendung: Setzt den Abstand zwischen der Oberkante der Kopfzeilenbox und dem oberen Blattrand.

Beispiel: `topmargin = 2cm`

textheight

Syntax: textheight = < *number* > < *unitkeyword* >;

Verwendung: Setzt die Höhe der Rumpfbox.

Beispiel: `textheight = 19cm`

textwidth

Syntax: textwidth = < *number* > < *unitkeyword* >;

Verwendung: Setzt die Breite der Rumpfbox.

Beispiel: `textwidth = 16cm`

3.3 Absatzlayout

Den meisten Schlüsselwortdefinitionen für das Absatzlayout, die in der ATTR-Umgebung definiert werden, wird ein numerischer Wert (< *number* >) gefolgt von einer Maßeinheit (< *unitkeyword* >) zugewiesen. Die nachfolgenden Maße stehen zur Verfügung:

< *unitkeyword* >	**Bedeutung**
mm	Millimeter
cm	Zentimeter
in	Zoll
pt	Point
em	zum jeweiligen Zeichensatz relatives Maß für die Breite eines M
ex	zum jeweiligen Zeichensatz relatives Maß für die Höhe eines X

---

alignment

Syntax: alignment = < *argument* >;

Verwendung: Gibt die Formatierausrichtung an. < *argument* > kann folgende Werte annehmen:

< *argument* >	**Bedeutung**
left	linksbündigen Flattersatz
right	rechtsbündigen Flattersatz
centered	zentrierten Satz
justified	Blocksatz

Beispiel: `alignment = justified`

---

baselinediff

Syntax: baselinediff = < *number* > < *unitkeyword* >;

Verwendung: Setzt den vertikalen Abstand zwischen zwei Grundlinien.

Beispiel: `baselinediff = 1.8ex`

---

hspace

Syntax: hspace = < *number* > < *unitkeyword* >;

Verwendung: Erzeugt einen horizontalen Leerraum.

Beispiel: `hspace = 3cm`

---

lineskiplimit

Syntax: lineskiplimit = < *number* > < *unitkeyword* >;

Verwendung: Setzt den minimalen vertikalen Abstand zwischen zwei Buchstabenboxen.

Beispiel: `lineskiplimit = .5ex`

leftindent

Syntax: leftindent $= < number > < unitkeyword >$;

Verwendung: Erzeugt einen linksseitigen Einzug.

Beispiel: `leftindent = 24mm`

rightindent

Syntax: rightindent $= < number > < unitkeyword >$;

Verwendung: Erzeugt einen rechtsseitigen Einzug.

Beispiel: `rightindent = 0.5in`

parfillskip

Syntax: parfillskip $= < number > < unitkeyword >$;

Verwendung: Setzt den Einzug der letzten Zeile eines Absatzes.

Beispiel: `parfillskip = 2em`

parindent

Syntax: parindent $= < number > < unitkeyword >$;

Verwendung: Setzt den Einzug der ersten Zeile eines Absatzes.

Beispiel: `parindent = 2pt`

postparsep

Syntax: postparsep $= < number > < unitkeyword >$;

Verwendung: Setzt den Abstand zwischen dem aktuellen und dem nachfolgenden Absatz.

Beispiel: `postparsep = 2ex`

preparsep

Syntax: preparsep $= < number > < unitkeyword >$;

Verwendung: Setzt den Abstand zwischen dem vorhergehenden und dem aktuellen Absatz.

Beispiel: `preparsep = 1.5ex`

3.4 Zeichensätze

font

Syntax: font = < *fontname* >;

Verwendung: Mit font wird der ab sofort gültige Zeichensatz gewählt. < *fontname* > hängt von den Zeichensatzdateien ab, auf die das System Zugriff hat. Im folgenden Beispiel wird eingestellt, daß Text durch den Zeichensatz Times abgebildet wird:

Beispiel:

```
font = Times;
```

Siehe auch: fontsize, fontstyle.

fontstyle

Syntax: fontstyle = < *style* >[,< *style* >,...];

Verwendung: Definiert den Schriftschnitt für den aktuellen Zeichensatz. Mögliche Angaben für < *style* > sind:

< *style* >	**Bedeutung**
bold	fett
italic	kursiv
plain	ohne Attribute
strikethrough	durchgestrichen
subscript	tiefergestellt
superscript	hochgestellt
underline	unterstrichen

Ein Zeichensatz kann mehrere Schriftschnitte gleichzeitig bekommen. Die gewünschten Schnitte werden einfach durch Komma getrennt aufgeführt. Die Kombination von subscript mit superscript ist nicht möglich. Sollten die beiden Schlüsselworte doch beide verwendet werden, gilt das jeweils zuletzt aufgeführte.

Beispiel: $\boxed{1}$ In diesem Beispiel wird der aktuelle Zeichensatz auf Fettdruck umgeschaltet:

```
fontstyle = bold;
```

$\boxed{2}$ Das folgende Beispiel schaltet auf hochgestellte kursive Schrift um. Man beachte die Vermischung von super- und subscript!

```
fontstyle = italic, subscript, superscript;
```

Siehe auch: font, fontsize.

fontsize

Syntax: fontsize = < *size* >< *unit* >;

Verwendung: Durch fontsize wird die Schriftgröße eines Zeichensatzes eingestellt. Die Größenangabe besteht dabei immer aus einer numerischen Größe < *size* > und einer Einheit < *unit* >. Mögliche Einheiten sind:

< ***unitkeyword*** >	**Bedeutung**
mm	Millimeter
cm	Zentimeter
in	Zoll
pt	Point
em	zum jeweiligen Zeichensatz relatives Maß für die Breite eines M
ex	zum jeweiligen Zeichensatz relatives Maß für die Höhe eines X

Beispiel: `fontsize = 12pt;`

Siehe auch: font, fontstyle.

3.5 Zähler

Für die Beispiele in diesem Abschnitt sei die logische Struktur eines Textes gegeben als:

```
<!DOCTYPE BEISPIEL[
<!ELEMENT Body (Kapitel+,Nachtrag)>
<!ELEMENT Kapitel(KapUeberschrift,SubKapitel*,Paragraph*)>
<!ELEMENT SubKapitel(SubKapitelUeberschrift,Paragarph*)>
    .
    .
    .
]>
```

Die Präsentationsregeln für einen Unterkapitelzähler sehen dann so aus:

```
          .
          .
!Unterkapitelzaehler {
                     TYPE { counter }
                     ATTR {
                            numstart = ?Kapitel;
                            numincrement = ?SubKapitel;
                            numstartvalue = 0;
                            numsymbol = arabic;
                                        .
                                        .
                                        .
                          }
                    };
          .
          .
```

Die Nummerierung startet und wirkt innerhalb des logischen Objektes Kapitel. Der Zähler wird mit Null initialisiert und bei jedem Auftreten eines logischen Objektes SubKapitel um eins erhöht. Die Ausgabe des Zählerstandes erfolgt jeweils in arabischen Zahlen.

---

numstart

Syntax: numstart = < *log_object* >

Verwendung: Dient zur Bestimmung des logischen Objektes, nach dessen Auftreten der Zähler starten soll. D.h. im Beispiel wird der Zähler Kapitelweise zurückgesetzt.

Beispiel: `numstart = ?Kapitel`

---

numstartvalue

Syntax: numstartvalue = < *number* >

Verwendung: Setzt den Anfangswert eines Zählers.

Beispiel: `numstartvalue = 0`

---

numincrement

Syntax: numincrement = < *log_object* >

Verwendung: Dient zur Bestimmung des logischen Objektes, bei dessen Auftreten der Zähler inkrementiert werden soll.

Beispiel: `numincrement = ?SubKapitel`

numsymbol

Syntax: numsymbol = < *argument* >

Verwendung: Dient zur Auswahl der Zeichen, mit denen die Nummerierung dargestellt werden soll. Das < *argument* > kann folgende Werte annehmen:

< *argument* >	**Bedeutung**
alph	Kleinbuchstaben
alphl	Großbuchstaben
arabic	arabische Zahlen
roman	kleine römische Zahlen
romanl	große römische Zahlen

Beispiel: `numsymbol = arabic`

3.6 Verzeichnisse

groupflag

Syntax: groupflag = true | false;

Verwendung: Dieser Parameter kann im Attribut-Block von Verzeichnissen verwendet werden. Der groupflag Schalter ermöglicht das Gruppieren von Verzeichniseinträgen, d.h. tritt ein Verzeichniseintrag mehrfach auf, wird er nur einmalig ausgegeben. Die Seitenzahlen werden durch Komma getrennt angegeben.

Beispiel: $\boxed{1}$

```
groupflag = true;
order = ascending;
```

erzeugt folgendes Verzeichnis:

Massenspeicher	43
Messebau	2, 17, 31

$\boxed{2}$

```
groupflag = false;
order = page;
```

erzeugt folgendes Verzeichnis:

Messebau	2
Messebau	17
Messebau	31
Massenspeicher	43

linkobj

Syntax: linkobj = < *object* > | nil;

Verwendung: Der Parameter kann im Attributblock von Verzeichnissen verwendet werden. Das Linkobjekt gibt an, welche Objekte im Verzeichnis aufgeführt werden sollen. < *object* > kann ein logischer oder physikalischer Objektbezeichner sein.

Beispiel:

```
linkobj = Ueberschrift;
```

erzeugt folgendes Verzeichnis:

1. Einleitung	1
2. Hauptteil	2
3. Schlußsatz	9

nextobj

Syntax: nextobj = < *object* > | nil;

Verwendung: Der Parameter kann im Attribut-Block von Verzeichnissen verwendet werden. Das Nextobejekt gibt bei verschachtelten Verzeichnissen an, welches Objekt folgen soll. Stellt der zu definierende Eintrag die unterste Ebene dar, muß der Wert nil eingesetzt werden. < *object* > kann ein logischer oder physikalischer Objektbezeichner oder nil sein.

Beispiel:

```
!Inhaltsverzeichnis{
      TYPE{ catalog }
      ATTR{
            linkobj = ?Ueberschrift;
            nextobj = !Unterinhaltsverzeichnis;
            fontsize = 12pt;
          }
                   };

!Unterinhaltsverzeichnis{
      TYPE{ catalog }
      ATTR{
            linkobj = ?Unterueberschrift;
            nextobj = nil;
            fontsize = 8pt;
          }
                        };
```

Daraus resultiert folgendes Vezeichnis:

1. Einleitung	1
1.1. Untereinleitung 1	1
1.2. Untereinleitung 2	2
2. Hauptteil	4
2.1. Grober Überblick	4
2.2. Kleiner Überblick	6

order

Syntax: order = ascending | descending | page;

Verwendung: Der Parameter wird im Attribut-Block von Verzeichnisobjekten verwendet. ascending gibt eine aufsteigende Sortierreihenfolge der Verzeichniseinträge an. Mit descending wird die absteigende Sortierfolge definiert. Wir der Wert page eingesetzt, findet keine Sortierung statt, die Einträge werden in der Reihenfolge des Auftretens im Text dargestellt.

Beispiel: s. Befehl groupflag.

pagenumflag

Syntax: pagenumflag = true | false;

Verwendung: Der Parameter pagenumflag wird im Attribut-Block von Verzeichnisobjekten verwendet. Der Wert true schaltet die Ausgabe der Seitenzahl ein, mit dem Wert false wird die Ausgabe der Seitenzahlen unterdrückt. Dieser Parameter kann z.B. dazu verwendet werden, um in einem Inhaltsverzeichnis nur bei den Hauptüberschriften eine Seitenzahl anzugeben. In der Definition der Untereinträge wird der Wert auf false gesetzt.

Beispiel: 1

```
pagenumflag = true;
```

erzeugt folgendes Verzeichnis mit Seitennummern:

1. Einleitung	1
1.1. Untereinleitung 1	1
1.2. Untereinleitung 2	2
2. Hauptteil	4
2.1. Grober Überblick	4
2.2. Kleiner Überblick	6

2

```
pagenumflag = false;
```

erzeugt folgendes Verzeichnis ohne Seitennummern:

1. Einleitung
1.1. Untereinleitung 1
1.2. Untereinleitung 2
2. Hauptteil
2.1. Grober Überblick
2.2. Kleiner Überblick

itemizeflag

Syntax: itemizeflag = true | false;

Verwendung: Dieser Parameter wird im Attribut-Block von Listenobjekten verwendet. Ein Aufzählungsobjekt muß durch den Wert true gekennzeichnet werden.

Beispiel:

```
itemizeflag = true;
itemizesymbol = *;
```

erzeugt eine Liste wie folgt:

* Erster Listenpunkt
* Zweiter Listenpunkt

itemizesymbol

Syntax: itemizesymbol = $< character >$ | $< object >$;

Verwendung: Dieser Parameter wird im Attributblock von Listenobjekten verwendet. Das angegebene Zeichen wird jedem Aufzählungspunkt vorangestellt. Wird ein Objekt eingesetzt, dann wird entweder der dort abgelegte Text oder der Zählerstand (z.B. für Aufzählungen) verwendet. $< character >$ kann ein beliebiges Zeichen sein.

Beispiel: siehe itemizeflag.

pagenumpos

Syntax: pageunumpos = left | right;

Verwendung: Der Parameter wird im Attribut-Block von Verzeichnisobjekten verwendet. Er sagt aus, an welcher Stelle die Seitennummern erscheinen. left schreibt die Seitennummer direkt hinter die Überschrift, right an den rechten Seitenrand.

Beispiel:

```
pagenumpos = right
```

3.7 Noten

notesymbol

Syntax: notesymbol = < *character* >;

Verwendung: Dieser Parameter wird im Attribut-Block von Referenzobjekten verwendet. Das angegebene Zeichen wird als Referenzsymbol im Text eingefügt. Kommen mehrere Referenzen in einem Textabschnitt vor, wird die Zahl der Zeichen vervielfacht. < *character* > kann ein beliebiges Zeichen sein.

Beispiel:

```
notesymbol = *;
notesymbolstart = kapitel;
notestartvalue = 1;
```

ergibt folgendes Notenformat:

Dieser Text hat* keine Anmerkungen** am Rand.

notesymbolstart

Syntax: notesymbolstart = < *object* >;

Verwendung: Dieser Parameter wird im Attribut-Block von Referenzobjekten verwendet. Tritt das angegebene Objekt im Text auf, wird die Notennummerierung auf den Startwert zurückgesetzt. < *object* > stellt ein logisches oder physikalisches Objekt dar.

Beispiel:

```
notesymbolstart=2;
```

erzeugt als erste Note:

Die ist ein Notentesttext** zu Test von Noten.

notestartvalue

Syntax: notestartvalue = < *number* >;

Verwendung: Dieser Parameter wird im Attribut-Block von Referenzsymbolen verwendet. Er gibt den Startwert für die Notennummerierung an.

Beispiel:

```
notesymbolstart=2;
```

erzeugt folgende erste Note:

Die ist ein Notentesttext** zu Test von Noten.

Index