



Projekt STAT+
Teilprojekt

STF - Statistisches Tabellenformat StML-Zusatz

Robert Handler
Alexander Bertsch
10. März 2005
Version 1.5

Copyright Statistik Austria

Das Produkt und die darin enthaltenen Daten sind urheberrechtlich geschützt. Alle Rechte sind der Statistik Österreich vorbehalten. Die Vervielfältigung und Verbreitung der Daten sowie deren kommerzielle Nutzung ist ohne deren vorherige schriftliche Zustimmung nicht gestattet. Weiters ist untersagt, die Daten ohne vorherige schriftliche Zustimmung der STATISTIK AUSTRIA ins Internet zu stellen, und zwar auch bei unentgeltlicher Verbreitung. Eine zulässige Weiterverwendung ist jedenfalls nur mit korrekter Quellenangabe "Statistik Austria" gestattet.

| | | |
|----------|---|----------|
| 1 | ALLGEMEIN | 4 |
| 2 | REGELN | 4 |
| 2.1 | REIHENFOLGE UM DEN TEXT EINES ELEMENTES ZU FINDEN | 4 |
| 2.2 | REIHENFOLGE DER STYLES | 4 |
| 2.3 | SPRACHAUSZEICHNUNGEN..... | 5 |
| 2.4 | BEZEICHNUNG DER ID'S | 5 |
| 3 | FAQ | 6 |
| 3.1 | WIE SOLL TEXT IN EINEM ELEMENT GESPEICHERT WERDEN. | 6 |
| 3.2 | WIE WIRD DER TEXT VON EINEM ELEMENT GEFUNDEN..... | 6 |
| 3.3 | WIE SOLLEN DIE ID'S VERGEBEN WERDEN..... | 7 |
| 4 | STYLETEMPLATES | 8 |
| 4.1 | LISTE | 8 |
| 4.2 | ALPHABETICAL ATTRIBUTE LIST | 18 |
| 4.3 | ALPHABETICAL ELEMENT LIST | 19 |

1 Allgemein

Dieses Dokument ist als zusätzliches Dokument zum Dokument „StML-Definition“ zu sehen. Es soll weitere Informationen, Regeln und Hinweise zum Format liefern.

Weitere Dokumente:

- StML-Definition (Definition der StML)
- Projekthandbuch (Projektspezifische Erklärungen)
- StML (Anforderungen und Ziele)
- STF – Statistisches Tabellenformat (Grundsätzliche Überlegungen, Tabellendefinitionen)

2 Regeln

2.1 Reihenfolge um den Text eines Elementes zu finden

Reihenfolge (1 hat die niedrigste Priorität):

1. Der Text vom referenzierten Element (falls vorhanden). Der Text des referenzierten Elementes wird wieder entsprechend dieser Reihenfolge ermittelt.
2. Der Text vom Element selbst.
3. Der Text von ausgewerteten Kindelementen. (bei mehreren "CT" Elementen, wird der Text aus allen Elementen mit gleicher Sprachauszeichnung zusammengesetzt.)
4. Der alternative Text (Metainformation mit type "alternatively")

2.2 Reihenfolge der Styles

Die Ermittlung des anzuwendenden Styles ist als „überschreibende Vereinigung“ mehrerer Styles zu sehen. Gibt es einen Style, der mit dem Namen „default“ gekennzeichnet ist, dann ist dieser Style die Grundlage für die restlichen Styles.

Bsp.:

- Hat die Tabelle einen Style, der aussagt, dass die Schriftfarbe rot ist, allerdings die Zeile einen Style für eine grüne Schriftfarbe, dann muss die Schriftfarbe für diese Zeile grün sein. (Die Zeile hat bei der Reihenfolge eine höhere Priorität und überschreibt somit die Information der Tabelle.)
- Hat die Tabelle einen Style, der aussagt, dass die Schriftfarbe rot ist, und die Zeile einen Style, dass die Ausrichtung zentriert sein soll, dann muss für diese Zeile die Schriftfarbe rot und die Ausrichtung zentriert sein. (Vereinigung der Information.)

Reihenfolge für interne Styles (1 hat die niedrigste Priorität)

1. Tabelle
2. Identifizierte Teile
 - Überlagernde Bereiche sind geordnet vom Größeren zum Kleineren Bereich (header, stub,...)
 - Level geordnet (stub, stub mit level 2,...)
3. Spalte
4. Zeile
5. Zelle
6. Style vom Element selbst.

Reihenfolge für externe Styles

Analog zu internen Styles. Wobei externe Styles abhängig vom Attribut „priority“ der Regel, die interne Verwendung ersetzen, oder als Style mit höherer Priorität eingestuft werden. Siehe „StyleTemplates“.

2.3 Sprachauszeichnungen

Textelemente („CT“) können mit einem Kennzeichen versehen werden, damit ersichtlich wird, dass der Text in einer anderen Sprache vorliegt.

Als „default“-Sprache ist Deutsch („de“) anzunehmen. Wird das Format in einer anderen Sprache erstellt, so ist das Element „Language“ mit der entsprechenden Sprachauszeichnung zu füllen. Analog dazu gibt es ein „Language“ Element pro Tabelle. Einzelne Abweichungen davon können mit dem Attribut „xml:lang“ für ein „CT“-Element definiert werden.

Reihenfolge (1 hat die niedrigste Priorität)

1. "default"-Sprache Deutsch
2. "Language"-Element des Formates
3. "Language"-Element der Tabelle
4. Attribut „xml:lang“ des „CT“-Elementes

2.4 Bezeichnung der ID's

Die Bezeichnungen der ID's sollten möglichst kurz und „sprechend“ gewählt werden und müssen mit folgenden Startbuchstaben beginnen:

| Bereich | Id-Kennung |
|------------------|------------|
| Description | D... |
| MetalInformation | M... |
| Unit | U... |
| FactInformation | F... |
| Styles | S... |
| Identification | I... |
| Classification | C... |

Der „Rest“ der ID kann frei nach belieben gewählt werden. Es ist allerdings ratsam soweit als möglich die ID nicht nur mit einer fortlaufenden Nr. sondern auch mit zusätzlich Informationen zu füllen.

Bsp.: Eine Klassifikation, wobei der externe Schlüssel zusätzlich verwendet wird.

```
<Classification id="CW521" key="W521">
```

```
...
```

```
</Classification>
```

Bsp.: Definition einer Zeitreihe aus der ISIS

```
<Classification id="CA10" key="A10" type="time" version="1" level="1">
```

```
  <Name id="CA10.name">Zeitreihe</Name>
```

```
  <Value id="CA10.0203" key="0203">
```

```
    <ValData>Februar 2003</ValData>
```

```
  </Value>
```

```
</Classification>
```

3 FAQ

3.1 *Wie soll Text in einem Element gespeichert werden.*

Es gibt mehrere Möglichkeiten, wie ein Text zu einem Element gespeichert werden kann. Prinzipiell sollte der Text immer innerhalb eines „CT“-Element gespeichert werden (Variante B). Es gibt allerdings auch eine „schlanke“ Möglichkeit, wie Variante A zeigt. Einige Elemente besitzen die Möglichkeit andere bekannte Datentypen (Int,...) zu speichern. Diese Möglichkeit muss genutzt werden, um später den Datentyp wieder zu erkennen (Variante C).

Variante A

Der Text wird direkt dem Element hinzugefügt.

```
<Description>
  <Title>Der Text vom Titel</Title>
</Description>
```

Variante B

Soll der Text Abschnitte darstellen, Sprachauszeichnungen oder Links besitzen, so muss der Text innerhalb eines „CT“-Elementes eingefasst werden.

Bsp.: Text mit einem Paragraphen

```
<Description>
  <Title>
    <CT xsi:type="Paragraph">
      <Txt>Der Absatz</Txt>
    <CT>
  </Title>
</Description>
```

Bsp.: Text mit Sprachauszeichnung und Link

```
<Description>
  <Title>
    <CT xml:lang="de-at">
      <Txt>Ein österreichischer Text</Txt>
      <IntLink ref="Text2"/>
    <CT>
  </Title>
</Description>
```

Variante C

Eine Zahl innerhalb eines Elementes

```
<Fact ...>
  <Int>12</Int>
</Fact>
```

3.2 *Wie wird der Text von einem Element gefunden.*

Der Text wird entsprechend den definierten Regeln in „Reihenfolge um den Text eines Elementes zu finden“ gefunden.

Gibt es innerhalb eines Kindelementes mehrere „CT“ Elemente mit gleicher Sprachauszeichnung, dann wird der Text aus diesen Elementen gebildet.

Bsp.: Der gefundene Text lautet: „Ein Text aufgeteilt in mehreren Blöcken“

```
<Title>
  <CT>
```

```
<Txt>Ein Text</Txt>
<CT>
<CT>
  <Txt> aufgeteilt in mehreren </Txt>
<CT>
<CT>
  <Txt>Blöcken</Txt>
<CT>
</Title>
```

3.3 *Wie sollen die ID's vergeben werden.*

Die Bezeichnungen der ID's sollten möglichst kurz und „sprechend“ gewählt werden.

Bsp.: Id für eine Metainformation:

```
<Meta id="M14">
  ...
</Meta>
```

Bsp.: Id für einen identifizierten Kopfbereich:

```
<Identification id="Iheader">
  ...
</Identification>
```

4 StyleTemplates

In manchen Fällen wird es notwendig sein, die internen Styles von einem StML-Format extern zu überlagern. Für diesen Fall können „StyleTemplates“ definiert werden, welche eine Reihe von Regeln enthält um dies zu bewerkstelligen. Dies sollte die Regel sein, um einheitliches Aussehen zu gewährleisten.

Allgemein:

Es wird eine Regel definiert, die aussagt auf welchen Knoten vom ursprünglichen XML die Regel und somit der definierte Style angewendet werden soll. Siehe „Reihenfolge der Styles“.

Des Weiteren ist es durch diese „StyleTemplates“ auch möglich intern festgelegte Styles aufzuheben.

4.1 Liste

4.1.1 Element: StyleTemplate

Documentation:

Das Rootelement, fuer externe Stylevorlagen.

Properties:

Type: complexType

Potential Child Element:

| Name | Type | Default | Min Occurs | Max Occurs |
|----------------------------------|-------------|---------|------------|------------|
| 4.1.2 stl:Styles | complexType | | 1 | 1 |
| 4.1.3 Rules | complexType | | 1 | 1 |

Tree:

- [4.1.1 StyleTemplate](#)
 - Sequence
 - [4.1.2 stl:Styles](#)
 - [4.1.3 Rules](#)

4.1.2 Element: stl:Styles

Documentation:

Container, um verschiedene Styles aufnehmen zu koennen.

Properties:

Type: complexType

Potential Child Element:

| Name | Type | Default | Min Occurs | Max Occurs |
|---------------------------------|-------------|---------|------------|------------|
| 4.1.4 stl:Style | complexType | | 1 | unbounded |

References:

| Used By |
|-------------------------------------|
| 4.1.1 StyleTemplate |

Tree:

- [4.1.2 stl:Styles](#)
 - Sequence
 - [4.1.4 stl:Style](#) (1 .. unbounded)

4.1.3 Element: Rules

Documentation:

Container, um die Regeln aufzunehmen, mit welcher die Styles angewendet werden.

Properties:

Type: complexType

Potential Child Element:

| Name | Type | Default | Min Occurs | Max Occurs |
|----------------------------|-----------|---------|------------|------------|
| 4.1.5 Rule | xs:string | | 0 | unbounded |

References:

| Used By |
|-------------------------------------|
| 4.1.1 StyleTemplate |

Tree:

- [4.1.3 Rules](#)
 - Sequence
 - [4.1.5 Rule](#) (0 .. unbounded)

4.1.4 Element: stl:Style

Documentation:

Definiert ein Aussehen.

Properties:

Type: complexType

Attribute:

| Name | Summary |
|------|---|
| id | Type: xs:ID Use: optional |
| name | Benennung für den Style. Type: xs:string Use: optional |

Potential Child Element:

| Name | Type | Default | Min Occurs | Max Occurs |
|--|-------------|---------|------------|------------|
| 4.1.6 stl:Borders | complexType | | 0 | 1 |
| 4.1.7 stl:Alignment | complexType | | 0 | 1 |
| 4.1.8 stl:Font | complexType | | 0 | 1 |
| 4.1.9 stl:Format | complexType | | 0 | 1 |
| 4.1.10 stl:Interior | complexType | | 0 | 1 |
| 4.1.11 stl:Description | | | 0 | 1 |

References:

| Used By |
|----------------------------------|
| 4.1.2 stl:Styles |

Tree:

- [4.1.4 stl:Style](#)
 - Sequence
 - [4.1.6 stl:Borders](#) (0 .. 1)
 - [4.1.7 stl:Alignment](#) (0 .. 1)
 - [4.1.8 stl:Font](#) (0 .. 1)
 - [4.1.9 stl:Format](#) (0 .. 1)
 - [4.1.10 stl:Interior](#) (0 .. 1)
 - [4.1.11 stl:Description](#) (0 .. 1)

4.1.5 Element: Rule

Documentation:

Definiert in XPATH Notation den Pfad fuer Knoten, auf das ein Style angewendet werden soll. Informationen ueber XPATH koennen unter "<http://www.w3.org/TR/xpath>" gefunden werden.

Properties:

Type: xs:string

Attribute:

| Name | Summary |
|----------|--|
| priority | Definiert, welche Prioritaet diese Regel hat. Type: xs:NMTOKEN Use: optional Default: normal Enumeration: Value: high Definiert, dass dieser Style hoehere Prioritaet hat, als jeder interne Style. Value: normal Definiert, dass dieser Style den internen ersetzt. |
| useStyle | Definiert, welcher Style fuer diese Regel verwendet werden kann. Wird das Key-Wort "none" verwendet, dann soll kein Style fuer diese Regel verwendet werden (Der interne Style soll entfernt werden). Type: xs:IDREF Use: optional |

References:

| Used By |
|-----------------------------|
| 4.1.3 Rules |

Examples:

Achtung: Es kommt immer auf die Prioritaet der Regel und auf das angewandte Element an, ob eine Regel im Programm auch beruecksichtigt wird.

```

...
  <stl:Style id="st1">
    <Font color="red">
  </stl:Style>
...

```

 Beschreibung: Alle Fussnoten sollen rot sein.

Source:

```
<Rule useStyle="st1">//Meta/@type="footnote"/CT</Rule>
```

Beschreibung: Der Titel soll eine rote Schrift haben. (Es kann sein, dass ein interner Style eine hoehere Prioritaet hat.)

```
<Rule useStyle="st1">//Title</Rule>
```

Beschreibung: Der Titel MUSS eine rote Schrift haben.

```
<Rule useStyle="st1" priority="high">//Title</Rule>
```

Beschreibung: Der Kopf der Tabelle soll eine rote Schrift haben.

Source:

```
<Rule useStyle="st1">//Identification type="header"</Rule>
```

Beschreibung: Jede zweite Zeile soll eine rote Schrift haben.

Source:

```
<Rule useStyle="st1">//TableView/Row[position() mod 2 = 0]</Rule>
```

 Beschreibung: Der interne Style fuer die Vorspalte soll nicht verwendet werden.

Source:

```
<Rule useStyle="none">//Identification type="stub"</Rule>
```

4.1.6 Element: **stl:Borders**

Documentation:

Container, um die Raender zu sammeln.

Properties:

Type: complexType

Potential Child Element:

| Name | Type | Default | Min Occurs | Max Occurs |
|-----------------------------------|-------------|---------|------------|------------|
| 4.1.12 stl:Border | complexType | | 1 | unbounded |

References:

| Used By |
|---------------------------------|
| 4.1.4 stl:Style |

Tree:

- [4.1.6 stl:Borders](#)
 - Sequence
 - [4.1.12 stl:Border](#) (1 .. unbounded)

4.1.7 Element: **stl:Alignment**

Documentation:

Definiert die Ausrichtung.

Properties:

Type: complexType

Attribute:

| Name | Summary |
|--------------|---|
| horizontal | Definiert die horizontale Ausrichtung. Type: xs:NMTOKEN Use: optional Enumeration: Value: center Zentriert Value: left Linksbuendig Value: right Rechtsbuendig |
| indent | Definiert, um wie viel der Text eingerueckt werden soll. Einheit = cm. Type: xs:float Use: optional minExclusive 0 |
| rotate | Definiert den Winkel, in welchem der Text geschrieben wird. 0° ist gueltig fuer eine Schrift, welche horizontal von links nach rechts geschrieben wird. 90° ist gueltig fuer eine Schrift, welche vertikal von unten nach oben geschrieben wird. Type: xs:int Use: optional Default: 0 minInclusive 0 maxInclusive 360 |
| vertical | Definiert die vertikale Ausrichtung. Type: xs:NMTOKEN Use: optional Enumeration: Value: bottom Unten Value: center Zentriert Value: top Oben |
| verticalText | Definiert, dass der Text buchstabenweise von oben nach unten dargestellt werden soll. Type: xs:boolean Use: optional |
| wrapText | Definiert, dass der Text umgebrochen werden darf. Type: xs:boolean Use: optional |

References:

| Used By |
|---------------------------------|
| 4.1.4 stl:Style |

4.1.8 Element: **stl:Font**

Documentation:

Definiert die Schriftart.

Properties:

Type: complexType

Attribute:

| Name | Summary |
|---------------|--|
| bold | Definiert, dass der Text fett geschrieben werden soll. Type: xs:boolean Use: optional |
| color | Definiert die Schriftfarbe. Die Werteangabe hat hexadezimal (0-9,A-F) zu erfolgen oder es handelt sich um die Angabe aus der nachfolgenden Liste: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow. Format = #RRGGBB Bsp.: #FFFFFF = Weiss Type: xs:string Use: optional Pattern: (#([0-9] [A-F]){6,6})(black) (blue) (cyan) (darkGray) (gray) (green) (lightGray) (magenta) (orange) (pink) (red) (white) (yellow) |
| family | Familie der Schriftart. Type: xs:string Use: optional |
| fontName | Name der Schriftart. Type: xs:string Use: optional |
| italic | Definiert, dass der Text kursiv geschrieben werden soll. Type: xs:boolean Use: optional |
| size | Definiert die Schriftgrosse. Einheit = cm. Type: xs:float Use: optional minExclusive 0 |
| underline | Definiert, dass unterstrichen werden soll. Type: xs:NMTOKEN Use: optional Enumeration: Value: double Doppelt Value: single Einfach |
| verticalAlign | Definiert, dass die vertikale Ausrichtung des Texts geaendert werden soll. Type: xs:NMTOKEN Use: optional Enumeration: Value: subscript Tiefgestellt Value: superscript Hochgestellt |

References:

| Used By |
|---------------------------------|
| 4.1.4 stl:Style |

4.1.9 Element: stl:Format**Documentation:**

Definiert das Format zur Darstellung.

Nachfolgend alle moeglichen "format" Attribut Pattern (entspricht der Java-Dokumentation zu DecimalFormat).

Patterns

DecimalFormat patterns have the following syntax:

Pattern:

PositivePattern

PositivePattern ; NegativePattern

PositivePattern:

Prefix_{opt} Number Suffix_{opt}

NegativePattern:

Prefix_{opt} Number Suffix_{opt}

Prefix:

```

        any Unicode characters except \uFFFE, \uFFFF, and special
characters
    Suffix:
        any Unicode characters except \uFFFE, \uFFFF, and special
characters
    Number:
        Integer Exponentopt
        Integer . Fraction Exponentopt
    Integer:
        MinimumInteger
        #
        # Integer
        # , Integer
    MinimumInteger:
        0
        0 MinimumInteger
        0 , MinimumInteger
    Fraction:
        MinimumFractionopt OptionalFractionopt
    MinimumFraction:
        0 MinimumFractionopt
    OptionalFraction:
        # OptionalFractionopt
    Exponent:
        E MinimumExponent
    MinimumExponent:
        0 MinimumExponentopt

```

A DecimalFormat pattern contains a positive and negative subpattern, for example, "#,##0.00;(#,##0.00)". Each subpattern has a prefix, numeric part, and suffix. The negative subpattern is optional; if absent, then the positive subpattern prefixed with the localized minus sign (code>'-' in most locales) is used as the negative subpattern. That is, "0.00" alone is equivalent to "0.00;-0.00". If there is an explicit negative subpattern, it serves only to specify the negative prefix and suffix; the number of digits, minimal digits, and other characteristics are all the same as the positive pattern. That means that "#,##0.0#;(#)" produces precisely the same behavior as "#,##0.0#;(#,##0.0#)".

The prefixes, suffixes, and various symbols used for infinity, digits, thousands separators, decimal separators, etc. may be set to arbitrary values, and they will appear properly during formatting. However, care must be taken that the symbols and strings do not conflict, or parsing will be unreliable. For example, either the positive and negative prefixes or the suffixes must be distinct for DecimalFormat.parse() to be able to distinguish positive from negative values. (If they are identical, then DecimalFormat will behave as if no negative subpattern was specified.) Another example is that the decimal separator and thousands separator should be distinct characters, or parsing will be impossible.

The grouping separator is commonly used for thousands, but in some countries it separates ten-thousands. The grouping size is a constant number of digits between the grouping characters, such as 3 for 100,000,000 or 4 for 1,0000,0000. If you supply a pattern with multiple grouping characters, the interval between the last one and the end of the integer is the one that is used. So "#,##,###,####,#####" == "#####,#####" == "##,####,#####".

Special Pattern Characters

Many characters in a pattern are taken literally; they are matched during parsing and output unchanged during formatting. Special characters, on the other hand, stand for other characters, strings, or classes of characters. They must be quoted, unless noted otherwise, if they are to appear in the prefix or suffix as literals.

The characters listed here are used in non-localized patterns. Localized patterns use the corresponding characters taken from this formatter's DecimalFormatSymbols object instead, and these characters lose their special status. Two exceptions are the currency sign and quote, which are not localized.

| Symbol | Location | Localized? | Meaning |
|---------------|---------------------|------------|--|
| 0 | Number | Yes | Digit |
| # | Number | Yes | Digit, zero shows as absent |
| . | Number | Yes | Decimal separator or monetary decimal separator |
| - | Number | Yes | Minus sign |
| , | Number | Yes | Grouping separator |
| E | Number | Yes | Separates mantissa and exponent in scientific notation. <i>Need not be quoted in prefix or suffix.</i> |
| ; | Subpattern boundary | Yes | Separates positive and negative subpatterns |
| % | Prefix or suffix | or Yes | Multiply by 100 and show as percentage |
| \u2030 | Prefix or suffix | or Yes | Multiply by 1000 and show as per mille |
| ¤ (\u00A4) | Prefix or suffix | or No | Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator. |
| ' | Prefix or suffix | or No | Used to quote special characters in a prefix or suffix, for example, "'###" formats 123 to "#123". To create a single quote itself, use two in a row: "# o'clock". |

Scientific Notation

Numbers in scientific notation are expressed as the product of a mantissa and a power of ten, for example, 1234 can be expressed as 1.234×10^3 . The mantissa is often in the range $1.0 \leq x < 10.0$, but it need not be. DecimalFormat can be instructed to format and parse scientific notation *only via a pattern*; there is currently no factory method that creates a scientific notation format. In a pattern, the exponent character immediately followed by one or more digit characters indicates scientific notation. Example: "0.###E0" formats the number 1234 as "1.234E3".

- The number of digit characters after the exponent character gives the minimum exponent digit count. There is no maximum. Negative exponents are formatted using the localized minus sign, *not* the prefix and suffix from the pattern. This allows patterns such as "0.###E0 m/s".
- The minimum and maximum number of integer digits are interpreted together:
 - If the maximum number of integer digits is greater than their minimum number and greater than 1, it forces the exponent to be a multiple of the maximum number of integer digits, and the minimum number of integer digits to be interpreted as 1. The most common use of this is to generate *engineering notation*, in which the exponent is a multiple of three, e.g., "##0.#####E0". Using this pattern, the number 12345 formats to "12.345E3", and 123456 formats to "123.456E3".
 - Otherwise, the minimum number of integer digits is achieved by adjusting the exponent. Example: 0.00123 formatted with "00.###E0" yields "12.3E-4".
- The number of significant digits in the mantissa is the sum of the *minimum integer* and *maximum fraction* digits, and is unaffected by the maximum integer digits. For example, 12345 formatted with "##0.##E0" is "12.3E3". To show all digits, set the significant digits count to zero. The number of significant digits does not affect parsing.
- Exponential patterns may not contain grouping separators.

Properties:

Type: complexType

Attribute:

| Name | Summary |
|------|---------|
|------|---------|

| | |
|--------------|---|
| fillWithDots | Definiert, dass die Zelle mit Punkten aufgefüllt werden soll. Bsp.: Niederoesterreich Type: xs:boolean Use: optional |
| format | Definiert das Format einer Zahl. Das Komma definiert die Stelle(n) fuer Tausender-Trennzeichen. Der Punkt definiert die Stelle des Dezimalzeichens. Bsp.: Zahl mit 2 Nachkommastellen und Tausender-Trennzeichen. "#,##0.00" Type: xs:string Use: optional |
| writeSpaced | Definiert, dass gesperrt geschrieben werden soll. Wobei die Anzahl der Blanks zwischen den Buchstaben angegeben wird. Type: xs:int Use: optional minInclusive 1 |

References:

| Used By |
|---------------------------------|
| 4.1.4 stl:Style |

4.1.10 Element: stl:Interior**Documentation:**

Definiert das Fuellverhalten (Hintergrund).

Properties:

Type: complexType

Attribute:

| Name | Summary |
|---------|---|
| color | Definiert die Fuellfarbe. Die Werteangabe hat hexadezimal (0-9,A-F) zu erfolgen oder es handelt sich um die Angabe aus der nachfolgenden Liste: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow. Format = #RRGGBB Bsp.: #FFFFFF = Weiss Type: xs:string Use: optional Default: white Pattern: (#([0-9] [A-F]){6,6})(black) (blue) (cyan) (darkGray) (gray) (green) (lightGray) (magenta) (orange) (pink) (red) (white) (yellow) |
| pattern | Definiert, die Art der Farbfuellung. Type: xs:NMTOKEN Use: optional Default: solid Enumeration: Value: solid Vollstaendig |

References:

| Used By |
|---------------------------------|
| 4.1.4 stl:Style |

4.1.11 Element: stl:Description**Documentation:**

Erweiterte Beschreibung fuer diesen Style.

Properties:**References:**

| Used By |
|---------|
|---------|

[4.1.4 stl:Style](#)

4.1.12 Element: stl:Border

Documentation:

Definiert das Aussehen eines Randes.

Properties:

Type: complexType

Attribute:

| Name | Summary |
|-----------|--|
| lineStyle | <p>Enthaelt die Art der Linie. Type: xs:NMTOKEN Use: optional Default: continuous Enumeration: Value: continuous Durchgaengig Value: dash Strichliert Value: dot Punktiert Value: double doppelte Linie Value: none Linie unterdruecken</p> |
| position | <p>Enthaelt die Position des Rands. Type: xs:NMTOKEN Use: required Enumeration: Value: bottom Unten Value: left Links Value: right Rechts Value: top Oben</p> |
| weight | <p>Enthaelt die Staerke der Linie. Einheit = cm. Type: xs:float Use: optional maxInclusive 3 minInclusive 0</p> |

References:

| Used By |
|-----------------------------------|
| 4.1.6 stl:Borders |

4.2 *Alphabetical Attribute List*

| | |
|---------------|--|
| bold | <u>stl:Font</u> 4.1.8 |
| color | <u>stl:Font</u> 4.1.8 <u>stl:Interior</u> 4.1.10 |
| family | <u>stl:Font</u> 4.1.8 |
| fillWithDots | <u>stl:Format</u> 4.1.9 |
| fontName | <u>stl:Font</u> 4.1.8 |
| format | <u>stl:Format</u> 4.1.9 |
| horizontal | <u>stl:Alignment</u> 4.1.7 |
| id | <u>stl:Style</u> 4.1.4 |
| indent | <u>stl:Alignment</u> 4.1.7 |
| italic | <u>stl:Font</u> 4.1.8 |
| lineStyle | <u>stl:Border</u> 4.1.12 |
| name | <u>stl:Style</u> 4.1.4 |
| pattern | <u>stl:Interior</u> 4.1.10 |
| position | <u>stl:Border</u> 4.1.12 |
| priority | <u>Rule</u> 4.1.5 |
| rotate | <u>stl:Alignment</u> 4.1.7 |
| size | <u>stl:Font</u> 4.1.8 |
| underline | <u>stl:Font</u> 4.1.8 |
| useStyle | <u>Rule</u> 4.1.5 |
| vertical | <u>stl:Alignment</u> 4.1.7 |
| verticalAlign | <u>stl:Font</u> 4.1.8 |
| verticalText | <u>stl:Alignment</u> 4.1.7 |
| weight | <u>stl:Border</u> 4.1.12 |
| wrapText | <u>stl:Alignment</u> 4.1.7 |
| writeSpaced | <u>stl:Format</u> 4.1.9 |

4.3 *Alphabetical Element List*

R **Rule** 4.1.5 **Rules** 4.1.3

S **StyleTemplate** 4.1.1

s **stl:Alignment** 4.1.7 **stl:Border** 4.1.12 **stl:Borders** 4.1.6 **stl:Description** 4.1.11 **stl:Font**
4.1.8 **stl:Format** 4.1.9 **stl:Interior** 4.1.10 **stl:Style** 4.1.4 **stl:Styles** 4.1.2