

# Einführung in XSLT

Schulung für das  
Editionsprogramm Fraktionen im Deutschen Bundestag 1949 – 1990

04./05.10.2021

# Übersicht

- Was ist XSLT?
- Grundsätzliche Funktionsweise
- Aufbau einer Transformation
- Wichtige Elemente
- XSLT in oXygen
- Übungen

# XML: Vor- und Nachteile

- Vorteil: Die Trennung von Struktur, Inhalt und Aussehen.
- Nachteil: Die Trennung von Struktur, Inhalt und Aussehen.
- Problem: Wenn ein Ergebnis ausgegeben werden soll, muss man die Zielsprache auch kennen!
  - HTML, CSS, (JavaScript)
- Problem: Alles, was ein Ausgabedokument „leisten“ soll, muss in der Kodierung vorhanden sein.
  - z.B. Register, Verknüpfungen, Formatierungen

# Was ist XSLT?

- XSLT ist eine Sprache, um XML-Dokumente in verschiedene Zielformate zu transformieren
- XSLT ist ein W3C-Standard seit 1999, aktuell: XSLT 3.0
- *XSLT heißt Extensible Stylesheet Language Transformation*
- XSLT ist selbst XML
- XSL = XSLT + XSL-FO + XPath

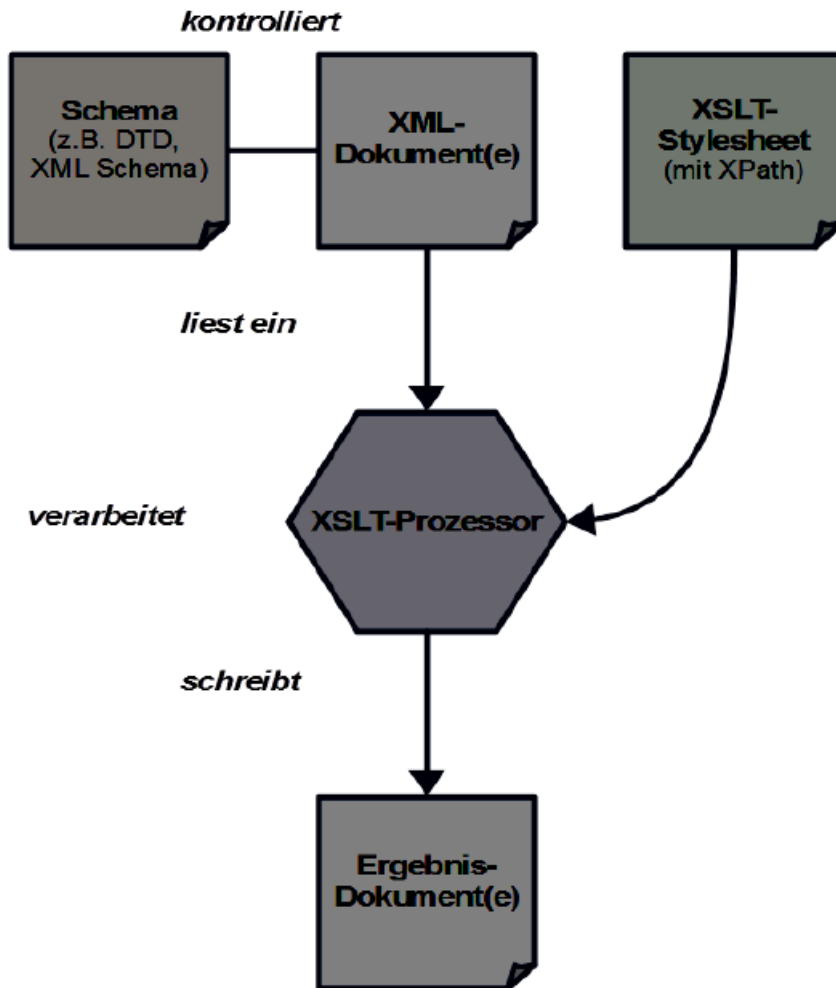
# Wozu braucht man XSLT?

Man braucht XSLT, um aus XML etwas (anderes) zu machen

- Reports
  - Z.B. Text
- Publikationen
  - Z.B. HTML/CSS
- Visualisierungen
  - Z.B. SVG, HTML/CSS/Javascript
- Exporte
  - Z.B. CSV, JSON, Datenbankformate
- XML
  - Daten verändern, anreichern, zusammenführen, extrahieren

... dies alles kann zusammenspielen!

# Grundsätzliche Funktionsweise



## Grundsätzliche Funktionsweise

- XSLT verwendet XPath
- XSLT wird von einem XSLT-Prozessor ausgeführt
- XML + XSLT -> Zieldokument(e)

# Grundsätzliche Funktionsweise

- Das Prinzip der drei Bäume:
  - Eingabebaum –Verarbeitungsbaum –Ausgabebaum
- Das Prinzip der Verarbeitung
  - Gehe durch den Eingabebaum und berücksichtige dabei die Anweisungen des Verarbeitungsbaums um den Ausgabebaum zu schreiben
- Das Prinzip der „Schablonen“
  - Wenn diese Schablone (ein XPath-Muster) passt, dann folge ihren Anweisungen

# Grundsätzliche Funktionsweise

Deutsch -> XSLT

Ich bin ein XML-Dokument

Ich bin ein XSLT-Stylesheet

Ich bin eine Schablone, ich passe auf ein Muster

Ich tue etwas, wenn das Muster im Ausgangsbaum gefunden wird. Meistens hole ich Informationen aus dem XML-Dokument, verarbeite sie und schreibe sie in das Zieldokument

Ich bin eine Schablone, ich passe auf ein Muster



# Strategien

## Pull

- Hole gezielt Daten und tue etwas damit  
-> typisch: for-each, for-each-group

## Push

- Verarbeite Daten, wenn sie Dir über den Weg laufen  
-> typisch: template, apply-templates, call-template

# Grundsätzliche Funktionsweise

Deutsch -> XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
  <xsl:template match="/">
    <h1>
      <xsl:value-of select="//titel" />
    </h1>
    <xsl:text>Hallo Welt</xsl:text>
  </xsl:template>

  <xsl:template match="lb"><br/></xsl:template>
</xsl:stylesheet>
```

# Ein XSLT-Stylesheet in oXygen

- Strg+n

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsl:stylesheet
```

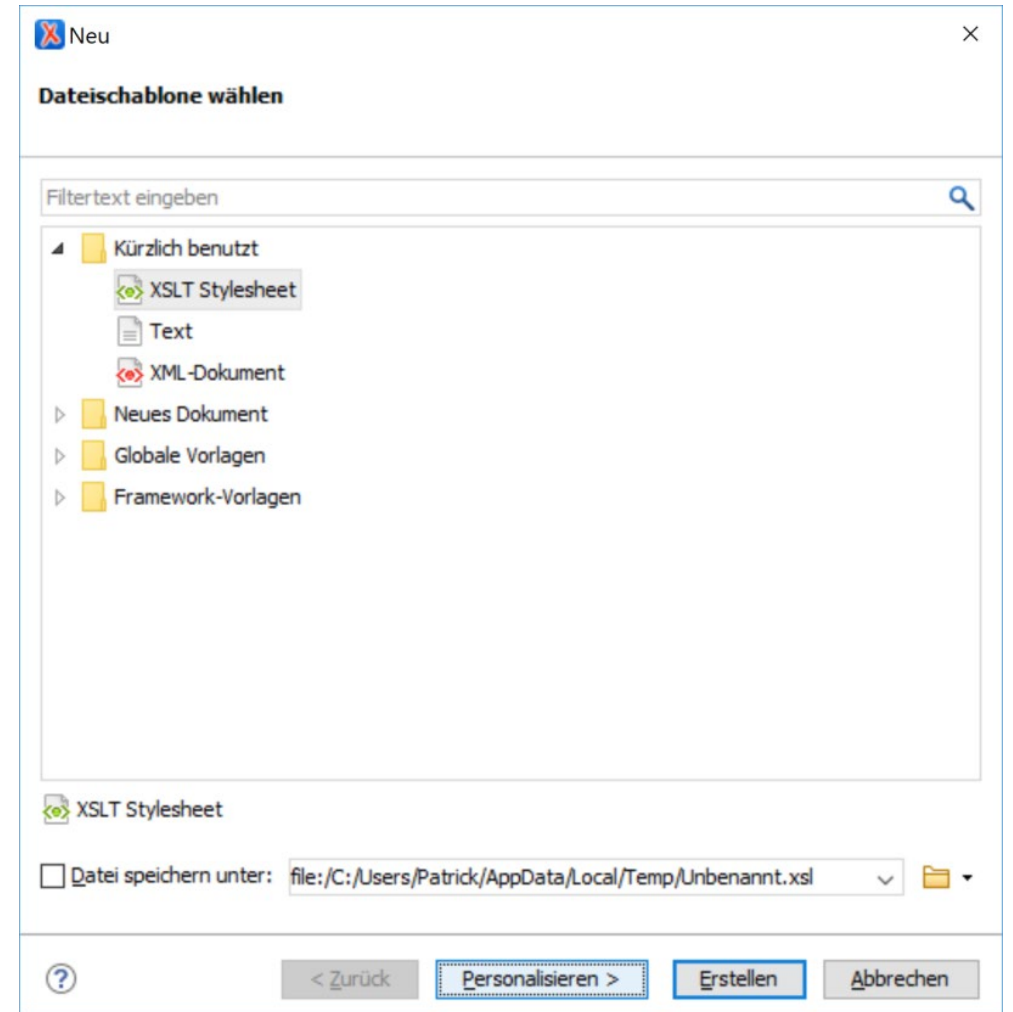
```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
exclude-result-prefixes="xs"
```

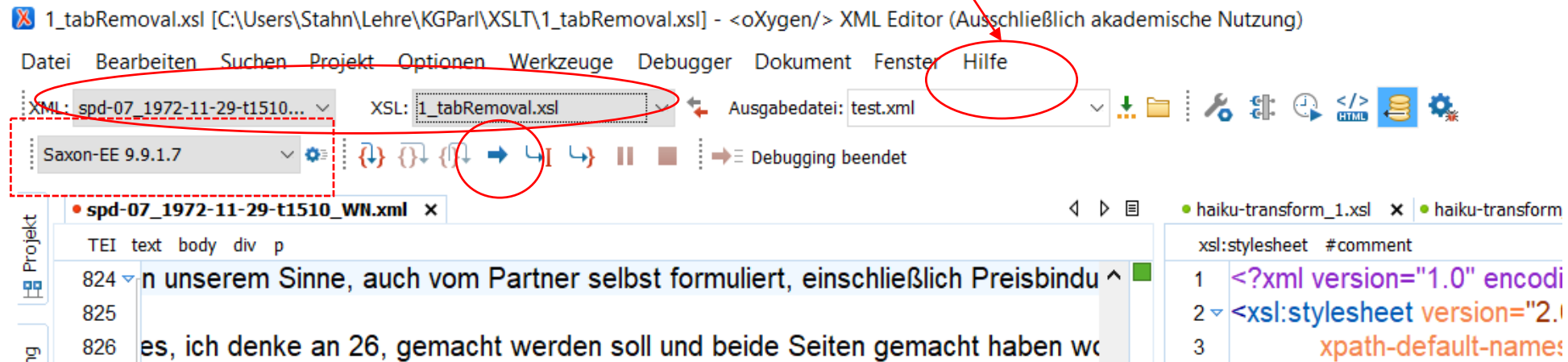
```
version="2.0">
```

```
</xsl:stylesheet>
```



# Eine Transformation in oXygen durchführen (Möglichkeit 1)

1. XSLT-Debugger öffnen: Fenster -> Perspektive öffnen -> Debugger auswählen (oder über Button in der Navigationsleiste)
2. Quelldokument auswählen: Strg + O -> in Ordner navigieren und entsprechende Datei auswählen
3. XSLT neu anlegen (s. vorige Folie) oder bestehende Datei öffnen
4. Dateien in der Werkzeugleiste auswählen, Ausgabedokument benennen
5. Transformation(Debugging) durchführen mit blauem Pfeil
6. Evtl. XSLT-Prozessor-Auswahl überprüfen
7. Ergebnis entweder direkt im Ausgabefenster überprüfen oder Datei öffnen



# Einige ~~Befehle~~Elemente

Vorab:

- Nochmal zu XSLT 1.0, XSLT 2.0, XSLT 3.0 ...
- Ca. 50 Elemente (in XSLT 2.0)
- Von denen Sie ca. 10-20 brauchen (um schon mal sehr weit zu kommen)
  - Erklärung: ein großer Teil der Magie findet in XPath statt
- Einfaches Tutorial (etwas veraltet):  
[https://www.w3schools.com/xml/xsl\\_intro.asp](https://www.w3schools.com/xml/xsl_intro.asp)
- Nicht gut (nur XSLT 1.0):  
[https://www.w3schools.com/xml/xsl\\_elementref.asp](https://www.w3schools.com/xml/xsl_elementref.asp)
- Eine etwas bessere Referenz (mit XSLT 2.0):  
<https://www.data2type.de/xml-xslt-xslfo/xslt/xslt-referenz/alphabetische-referenz-der-xslt-elemente/>

# Einige Elemente

<a href="#"><u>xsl:stylesheet</u></a>	ist das Wurzelement eines XSL-Dokumentes
<a href="#"><u>xsl:template</u></a>	enthält Instruktionen, deren Ausgabe direkt in das Ergebnisdokument eingetragen werden. Das Attribut match ( <i>XPath</i> ) sagt, auf welches Muster die Schablone passen soll
<a href="#"><u>xsl:apply-templates</u></a>	ruft das Template auf, dessen match-Attribut auf ein Kindknoten des aktuellen Kontextknotens passt.
<a href="#"><u>xsl:value-of</u></a>	wandelt eine im select-Attribut angegebene Sequenz in Werte um, die als Output in das Ergebnisdokument kopiert werden.

# Einige Elemente

<a href="#">xsl:for-each</a>	Tue für jedes ... etwas. Menge wird im Attribut select bestimmt.
<a href="#">xsl:for-each-group</a>	dient dazu, eine im select-Attribut angegebene Eingabesequenz anhand eines Kriteriums zu gruppieren.
<a href="#">xsl:sort</a>	sortiert eine (Knoten-)Sequenz anhand eines durch das select-Attribut ausgewählten Sortierkriteriums
<a href="#">xsl:if</a>	die klassische IF-Abfrage, ohne jedoch eine alternative ELSE-Möglichkeit zuzulassen.
<a href="#">xsl:choose</a>	enthält eine oder mehrere <a href="#">xsl:when</a> -und gegebenenfalls abschließend eine <a href="#">xsl:otherwise</a> -Instruktion.
<a href="#">xsl:variable</a>	definiert eine Variable, die mit einem Namen als Referenz versehen wird und der Werte, Strings oder ganzer Code zugeordnet werden kann.

# xsl:stylesheet

```
<xsl:stylesheet version="1.0"  
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
    ... Anweisungen hierher ...  
</xsl:stylesheet>
```

Wichtig: TEI-Dateien müssen mit TEI-Namespace verarbeitet werden!

```
<xsl:stylesheet version="1.0"  
xmlns:tei="http://www.tei-c.org/ns/1.0"  
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
</xsl:stylesheet>
```



# XSLT Beispiel

- XML: `<message>Hallo Welt!</message>`
- XSLT:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body>
      <h1><xsl:value-of select="message"/></h1>
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

- Der HTML-Namespace wird als default angenommen!

# Das Grundgerüst: Templates

```
<xsl:template match=" xpath-ausdruck ">
```

... Anweisungen ...

```
</xsl:template>
```

- Schablone als Anweisung, an welcher Stelle im XML-Dokument (match="XPath") etwas gemacht werden soll

# Ausgabe von Text

`<xsl:value-of select=" xpath-ausdruck" />`

- Wählt den **Wert** eines XPath-Ausdrucks, also **Text** eines Elements oder Attributs und fügt diesen in die Ausgabe ein.

`<h1> <xsl:value-of select="message" /> </h1>`

`<xsl:text> Schreibe diesen Text ... </xsl:text>`

# Schleifen

```
<xsl:for-each select=" xpath-ausdruck ">
```

```
...
```

```
</xsl:for-each>
```

- Beispiel: Wähle jeden Text (**//msItem**) und generiere eine Aufzählung (**<ul>**) der Titel (title):

```
<ul>
```

```
  <xsl:for-each select="//msItem">
```

```
    <li> <xsl:value-of select="title"/> </li>
```

```
  </xsl:for-each>
```

```
</ul>
```

# Bedingungen I

```
<xsl:if test=" xpath-ausdruck "> ... </xsl:if>
```

- Führt Anweisungen nur aus, wenn die Bedingung in @test erfüllt ist.
- Beispiel:

```
<xsl:for-each select="//book">  
  <xsl:if test="author = 'Cassiodor' ">  
    <li><xsl:value-of select="title"/></li>  
  </xsl:if>  
</xsl:for-each>
```

# Bedingungen II

```
<xsl:choose>  
  <xsl:when test=" xpath-ausdruck "> ... </xsl:when>  
  <xsl:otherwise></xsl:otherwise>  
</xsl:choose>
```

- Unterscheidet mehrere Fälle, in denen Anweisungen nur ausgeführt werden, wenn die Bedingung in @test erfüllt ist. Ansonsten werden die Anweisungen des xsl:otherwise ausgeführt.

# Sortieren

`<xsl:sort select=" xpath-ausdruck "/>`

- Sortiert die Elemente einer Schleife (nach dem ausgewählten Kriterium)
- Wird immer innerhalb von `<xsl:for-each>` oder `<xsl:apply-templates>` verwendet

- Beispiel:

`<ul>`

`<xsl:for-each select="//mslItem">`

`<xsl:sort select="author"/>`

`<li> <xsl:value-of select="title"/>`

`<xsl:text> von </xsl:text>`

`<xsl:value-of select="author"/> </li>`

`</xsl:for-each>`

`</ul>`

# Kopieren

`<xsl:copy/>`

- Kopiert den aktuellen Knoten inklusive Namespace
- Kopiert keine Attribute, Inhalte, Kindknoten

`<xsl:copy-of select="xpath-ausdruck"/>`

- Kopiert den ausgewählten Knoten inklusive aller Kindknoten und Attribute



# Variablen / Parameter

`<xsl:variable name="" />`

- wird einmalig festgelegt

`<xsl:param name="" />`

- kann „unterwegs“ verändert werden

# Übungen

## 0. Texttransformation 1

**Haiku\_tei.xml + haiku-transform\_1.xsl -> haiku\_<KÜRZEL>\_1.xml**

-> Push/Pull

## 1. Texttransformation 2

**Haiku\_tei.xml + haiku-transform\_2.xsl -> haiku\_<KÜRZEL>\_2.xml**

-> for-each, sort, call template

## 2. Text umformatieren

**spd-07\_1972-11-29-t1510\_WN.xml + 1\_tabRemoval.xsl -> spd\_<KÜRZEL>\_tabs.xml**

-> Identity Transform, normalize-space(), value-of

# Übungen II

## 3. Element entfernen

**spd\_tabs.xml + 2\_kapitaelchen.xsl** -> spd\_<KÜRZEL>\_kapitaelchen.xml

-> ggf. mit XPath Builder überprüfen, ob alle raus sind

## 4. Attribut einfügen

**spd\_kapitaelchen.xml + 3\_insertIds.xsl** -> spd\_<KÜRZEL>\_ids.xml

-> generate-id(), apply-templates

-> Vergleich copy-of / value-of

-> Namensraum für Zieldokument

# Übungen III

## 5. Attribut entfernen

**spd\_ids.xml + 4\_deletelds.xsl** -> spd\_<KÜRZEL>\_deletelds.xml

-> xsl:element, apply-templates, import

-> vgl. ohne select und mit select="@\\* | node()"

## 6. Einfach HTML-Ausgabe produzieren

**spd\_deletelds.xml + 5\_html.xsl** -> spd\_<KÜRZEL>\_html.html

-> apply-templates, value-of (PUSH vs. PULL)

-> ACHTUNG DATEIENDUNG

-> Namespaces

Probiert weitere Transformationen aus; Orientiert euch an <https://fraktionsprotokolle.de/handle/5184>

(z. B. Personen unterschiedlich gestalten abhängig von Rolle (<b>, <i>, ...) )

# Übungen IV

7. Teile der Datei extrahieren und in neue Datei schreiben

**/Protokolle/Fraktionen/spd/07/ + extractPersons.xsl** -> persons\_<KÜRZEL>.xml

-> ACHTUNG: extractPersons.xsl muss im selben Ordner liegen wie Protokolle-GitHub-Repository -> Copy&Paste; anschließend bitte wieder löschen

-> **Version 1:** result-document, for each, collection,

-> **Version 2:** if + message

# Quellen

- P. Sahle, „Verarbeitung von XML mit XSLT: Einführung“, IDE Winter School 2020, Wuppertal, [https://www.i-d-e.de/wp-content/uploads/2020/02/02\\_XML-Einf%C3%BChrung.pdf](https://www.i-d-e.de/wp-content/uploads/2020/02/02_XML-Einf%C3%BChrung.pdf) (abgerufen am 02.10.21)
- T. Schaßan, „Transformation von XML-Dokumenten“, IDE Summer School 2013, Chemnitz, <https://docplayer.org/8546485-Transformation-von-xml-dokumenten-ide-summerschool-2013-chemnitz.html> (abgerufen am 02.10.21)

# Transformationsszenarien

Wie bringt man XML, XSLT und den Prozessor zusammen?

a) Dieses XML-Dokument soll mit jenem XSLT-Dokument verarbeitet werden

b) Dieses XSLT-Dokument soll jenes XML-Dokument verarbeiten

c) Lieber Prozessor, verarbeite jenes XML-Dokument mit jenem XSLT-Dokument

-> oXygen „Transformationsszenario“

# oXygen-Transformationsszenario



Transformation-Szenarios konfigurieren (Strg+Umschalt+C)



Transformation-Szenarios anwenden (Strg+Umschalt+T)






# oXygen-Transformationsszenario




Neues Szenario

Name:

Speicherort: ☒ Projekt-Optionen ☐ Global-Optionen


XSLT FO-Prozessor **Ausgabedatei**

XML URL:    

XSL URL:    

[Mehr über \\${currentFileURL} lesen](#)

☐ Verwenden des Stylesheets der "xml-stylesheet" Deklaration

Transformator:  

Parameter (0)

Erweiterungen (0)

Zusätzliche XSLT-Stylesheets (0)

OK Abbrechen

Neues Szenario



Name:

Speicherort: ☒ Projekt-Optionen ☐ Global-Optionen

XSLT FO-Prozessor **Ausgabedatei**



Ausgabedatei

☐ Nach Datei fragen

☒ Datei speichern unter   

☒ In Browser-/Systemanwendung öffnen

☒ Gespeicherte Datei

☐ Andere Position   



☒ Im Editor öffnen

In der Ergebnis-Ansicht anzeigen als

☒ XML

☐ SVG

☐ XHTML

Grafik-URLs sind relativ zu dieser URL:   

Wählen Sie das, wenn Sie

OK Abbrechen