Nutzung von indirekten Formaten in LibreOffice für semantische Kennzeichnung

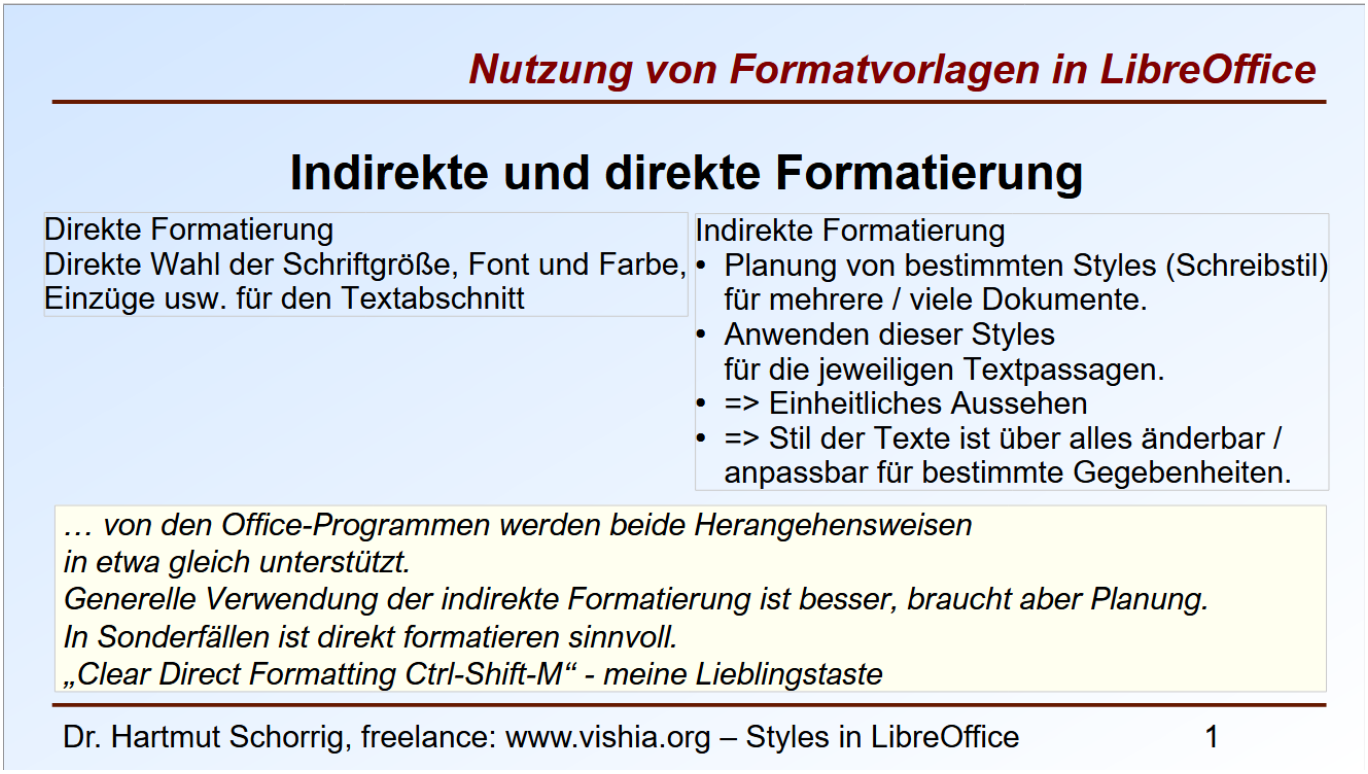
Vortrag auf Augsburger Linuxtag
2025-04-26

Inhalt

1 Intension

Alle Schreib- und Zeichenprogramme kennen die direkte Formatierung und die **indirekte Formatierung**. Was ist der Unterschied? Wenn ich etwas rot haben will, oder in einer bestimmten Schriftart, wähle ich für diesen Textteil die Farbe und Schriftart. Fertig. Das ist direkte Formatierung, praktisch für das Schreiben einfacher Texte.

Bei umfangreichen zu gestaltenden Texten im professionellen Bereich würde dies aber für notwendige Formatierungsänderungen (bspw. gesamter Text in anderer Schriftart, für eine bestimme Publikation) in Arbeit ausarten. Man könnte dies einer KI überlassen, die hat genügend Zeit, braucht aber etwas mehr Strom. Es geht auch einfacher und systematischer.



S1.png

Für solche Aufgabenstellungen wurde die Indirekte Formatierung erfunden, die ich selbst seit den Anfangszeiten von „Word für DOS“ etwa 1992 kenne.

Es ist so, dass beide Formatierungsmöglichkeiten nebeneinander und etwas in Konkurrenz bei der Office-Tool-Bedienung bzw. bei deren Bedienern stehen.

Doch es gibt noch einen anderen Aspekt, den der **Semantischen Kennzeichnung**.

2 Ein Zeichenprogramm für grafische Programmierung

Bei der Programmierung von Steuerungen (*„Embedded Control“*) stehen sich ebenfalls zwei Welten gegenüber, seit den 1990-ern oder noch etwas eher. Traditionell wird so etwas mit Code-Zeilen programmiert, bei Embedded Control (= Gerätesteuerungen) meist in C oder C++, für einfache Anwendungen wird oft auch Python favorisiert, oder auch andere und neuere Programmiersprachen, die sich aber nur schrittweise verbreiten, mit Halbwertszeiten von Jahrzehnten. Das ist traditionell.

Fast genau so lange gibt es die Grafische Programmierung. Der erste Standard für Automatisierungsprojekte ist aus den 1960-ern, die Norm IEC 61131 mit seinen Funktionsblock-

Diagrammen oder auch ursprünglich den grafischen Verschaltungen von Relais und Schaltern (sogenannte Latter-Graphic). Beides ist heute noch aktuell in Gebrauch.

Doch alle Tools für das Zeichnen dieser Diagramme, und deren Auswertung mit Zielsystem-Code-Generierung sind spezialisiert. Das vordergründige Problem sind dabei nicht unbedingt die Kosten für Lizenzen, sondern eher die Abhängigkeit vom Toolhersteller. Der Aufwand für die langjährige Pflege wächst, da man die „alten Tools“ weiter aufheben muss um Änderungen in Geräten ausführen zu können.

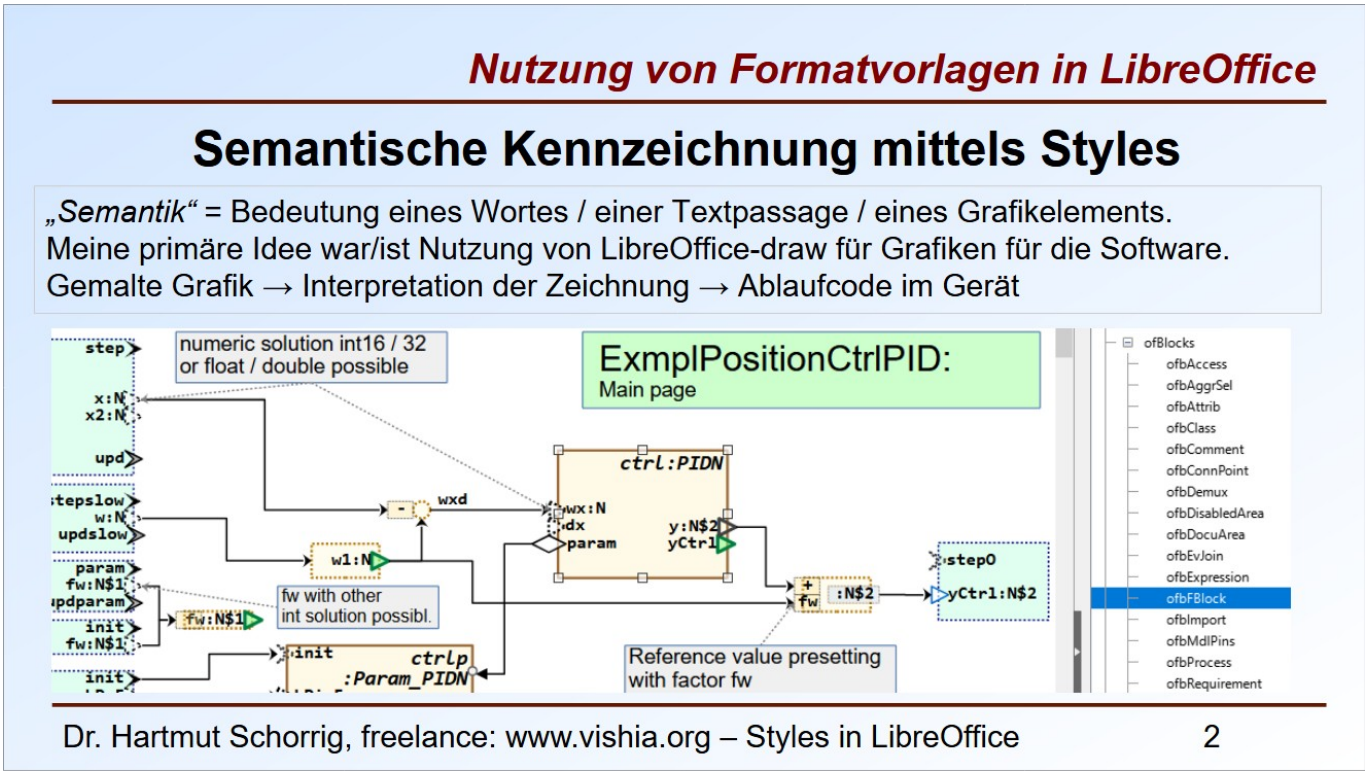
Dagegen kann bei der textuellen Programmierung jeder Texteditor verwendet werden, für einfache Korrekturen. Für komplexes Programmieren bringen die IDEs (Integrated Development Environment) eine Reihe von Unterstützungen, aber man hat immer die Rückfallebene, den einfachen Text. Die Tools zum Compilieren für das Zielsystem sind vergleichsweise mit weniger Aufwand aufzuheben und einzusetzen.

Dies mag eine der Gründe dafür sein, dass schon in den 1990-ern prognostiziert wurde, demnächst wird nur noch grafisch programmiert, die Prognose ist bis heute jedoch nicht eingetreten. Grafisch wird dann programmiert, wenn man sich auf ein Tool eingewöhnt hat. Der Toolwechsel ist schwierig.

Deshalb ist es interessant, ein weit verbreitetes und allgemeines Zeichentool zu verwenden, das noch dazu für die einfache Anwendung nichts kostet und sofort verfügbar ist. Daher bin ich mit einem Übersetzer der grafischen Programmierung beschäftigt. Zum Zeichen selbst wird LibreOffice Draw verwendet. Das Spezial-Tool reduziert sich nur noch auf den Übersetzer und Codegenerator, der vergleichsweise einfach zu pflegen ist, ebenfalls Open Source.

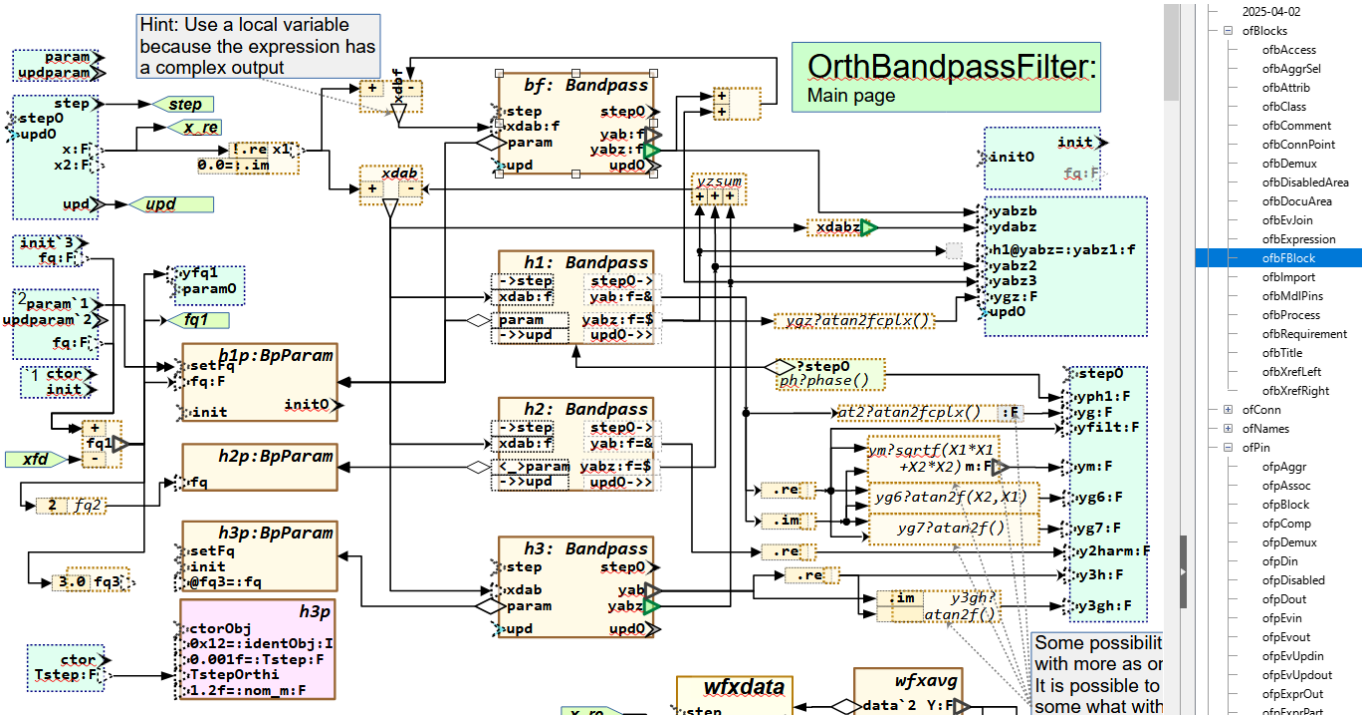
Doch wie kann man mit einem allgemeinen Zeichentool wie LibreOffice Draw Diagramme so zeichnen um danach die Zeichnung interpretieren zu können mit Code-Generierung? Muss man die Grafiken in ihren Strichstärken und Formen analysieren? Klingt danach dass die KI auch hier wieder unterstützend notwendig ist ? ??

Es geht einfacher! Die Styles (Formatvorlagen der Indirekten Formatierung) werden als semantische Kennzeichnung benutzt.



S2.png

3 Ein Grafikbeispiel mit Styles



odg/OrthBandpassFilter-Styles.odg.png

Im Bild sieht man den Block oben Mitte markiert, die Formatvorlagen-Auswahl rechts schnippt auf die zugewiesene Vorlage, den „Style“ **`ofpFBlock`**. Damit wird dieses Rechteckkästchen unabhängig von Farbe und Form als Rahmen eines „Funktionsblockes“ erkannt.

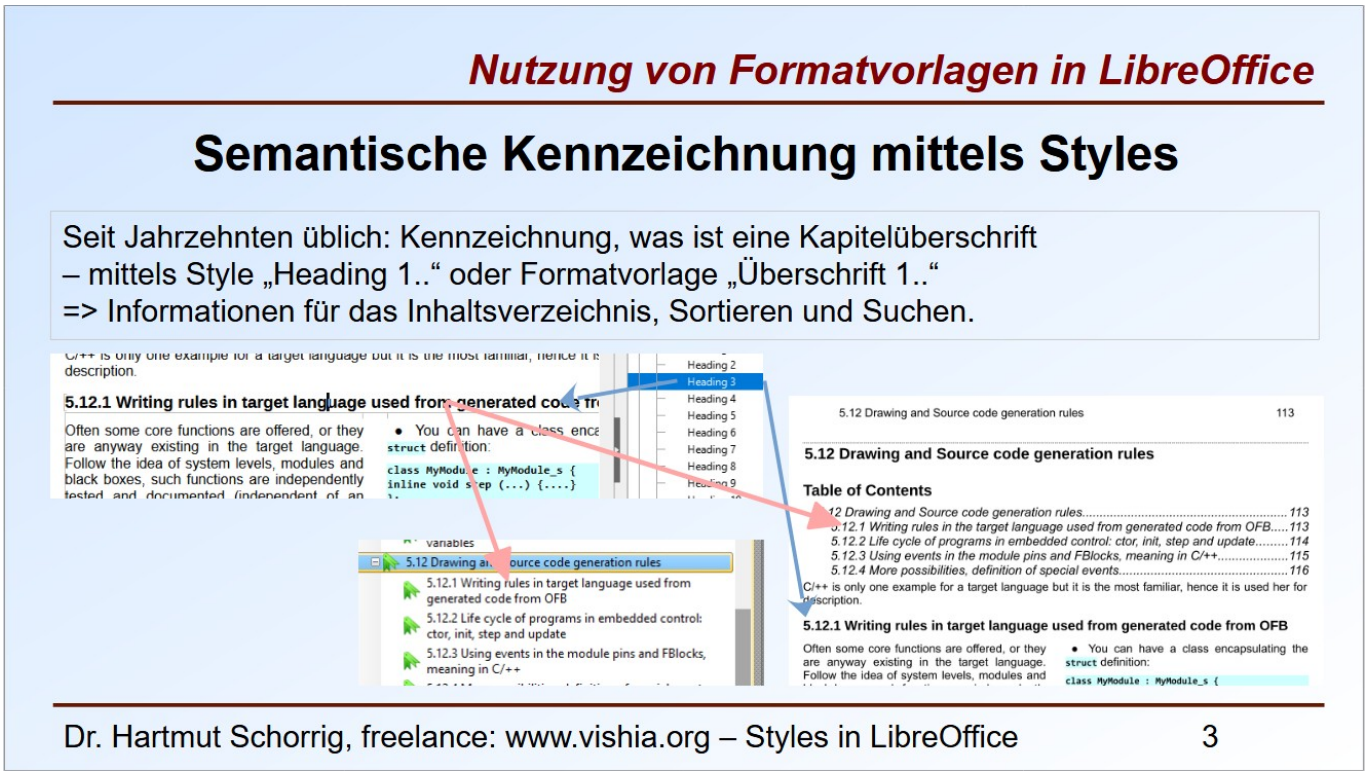
Die Pins (Anschlüsse) links und rechts haben eine bestimmte Form und Farbe. Diese ist jedoch ‚nur‘ für den Bediener bzw. für die anschauliche Darstellung des Diagramms. Die Art des Pins, bzw. dass es eben ein Pin ist, wird mit dem entsprechenden Style **`ofpPin…`** erkannt. Auch die Verbindungen zwischen den Funktionsblöcken und/oder Pins können einen Style haben wenn dies notwendig ist. Ansonsten bestimmt der Style der Pins die Art der Verbindung. „Verbinder“ oder „Connectors“ sind mit Klebepunkten (Glue points) an die entsprechenden Rechteck-Formen (Shapes) gebunden, so dass sie beim Verschieben sich mit ziehen. Der Zusammenhang des gezeichneten Funktionsblocks mit den gezeichneten Pins ist aufgrund der Position erkannt, wobei auch eine Gruppierung möglich ist, auch zum vereinfachten Schieben.

Und nun muss nur noch der Inhalt der Grafik gelesen und sortiert werden. Aufgrund der Style-Zuordnung an jedem Element ist das möglich und nicht zu komplex. Ob dies mit interner Programmierung, Zugriff innerhalb des LibreOffice zu den Interndaten evtl. unter Nutzung des Makrosystems gelingt, habe ich nicht getestet. Statt dessen werden die Daten aus der gespeicherten Form im odg-File gelesen. Intern ist dort ein XML-Format verwendet, dass als *„Open Document Format“* - Standard (ODF) langzeitstabil ist. Eine Grafik heute im aktuellem LibreOffice gespeichert kann dann sehr wahrscheinlich auch noch im Jahre 2045 oder 2055 bearbeitet und ausgelesen werden. Neue Versionen des ODF sollten abwärtskompatibel sein.

Der Übersetzer selbst ist ein nicht zu komplexes Java-Programm, Open Source, nachvollziehbar und aufgrund Langzeitversionierung bestimmter Java-Laufzeitumgebungen bzw. der Möglichkeit der Neuübersetzung ebenfalls noch nach Jahren verfügbar.

Weitere Details sind nicht Bestandteil dieser Dokumentation bzw. des Vortrages, können aber eingesehen werden in https://vishia.org/fbg/html/Videos_OFB_VishiaDiagrams.html .

4 Styles für semantische Kennzeichnung in Texten



S3.png

Für die Dokumentengliederung verwendete Kapitelüberschrift-Formatvorlagen haben nicht nur den Zweck, die Überschrift zu formattieren. Sie werden insbesondere benutzt, um den Text als Überschrift des jeweiligen Levels zu erkennen, um daraus das Inhaltsverzeichnis zu erstellen. Gleiches gilt für Bildunterschriften.

Die Indirekten Formattierungen können aber auch zur Text-Bedeutungs-Kennzeichnung verwendet werden. Als Beispiel sollen ein paar Bibelverse dienen:

1. Mose 11

Der Turmbau zu Babel

1 *Es hatte aber alle Welt einerlei Zunge und Sprache.* 2 *Als sie nun von Osten aufbrachen, fanden sie eine Ebene im Lande Schinar und wohnten daselbst.* 3 *Und sie sprachen untereinander:* ***Wohlauf, lasst uns Ziegel streichen und brennen!*** *– und nahmen Ziegel als Stein und Erdharz als Mörtel* 4 *und sprachen:* ***Wohlauf, lasst uns eine Stadt und einen Turm bauen, dessen Spitze bis an den Himmel reiche, dass wir uns einen Namen machen; denn wir werden sonst zerstreut über die ganze Erde.***

5 *Da fuhr der Herr hernieder, dass er sähe die Stadt und den Turm, die die Menschenkinder bauten.* 6 *Und der Herr sprach:* **Siehe, es ist einerlei Volk und einerlei Sprache unter ihnen allen und dies ist der Anfang ihres Tuns; nun wird ihnen nichts mehr verwehrt werden können von allem, was sie sich vorgenommen haben zu tun.** 7 **Wohlauf, lasst uns herniederfahren und dort ihre Sprache verwirren, dass keiner des andern Sprache verstehe!**

8 *So zerstreute sie der Herr von dort über die ganze Erde, dass sie aufhören mussten, die Stadt zu bauen.* 9 *Daher heißt ihr Name Babel, weil der Herr daselbst verwirrt hat aller Welt Sprache und sie von dort zerstreut hat über die ganze Erde.*

Der Text der Bibel als solches ist hervorgehoben durch Kennzeichnung mit dem Absatz-Style `BabelExample`. Damit kann schon mal aus einem größerem Dokument alle Bibelzitate nicht nur optisch erkannt, sondern auch technisch herausgelöst werden.

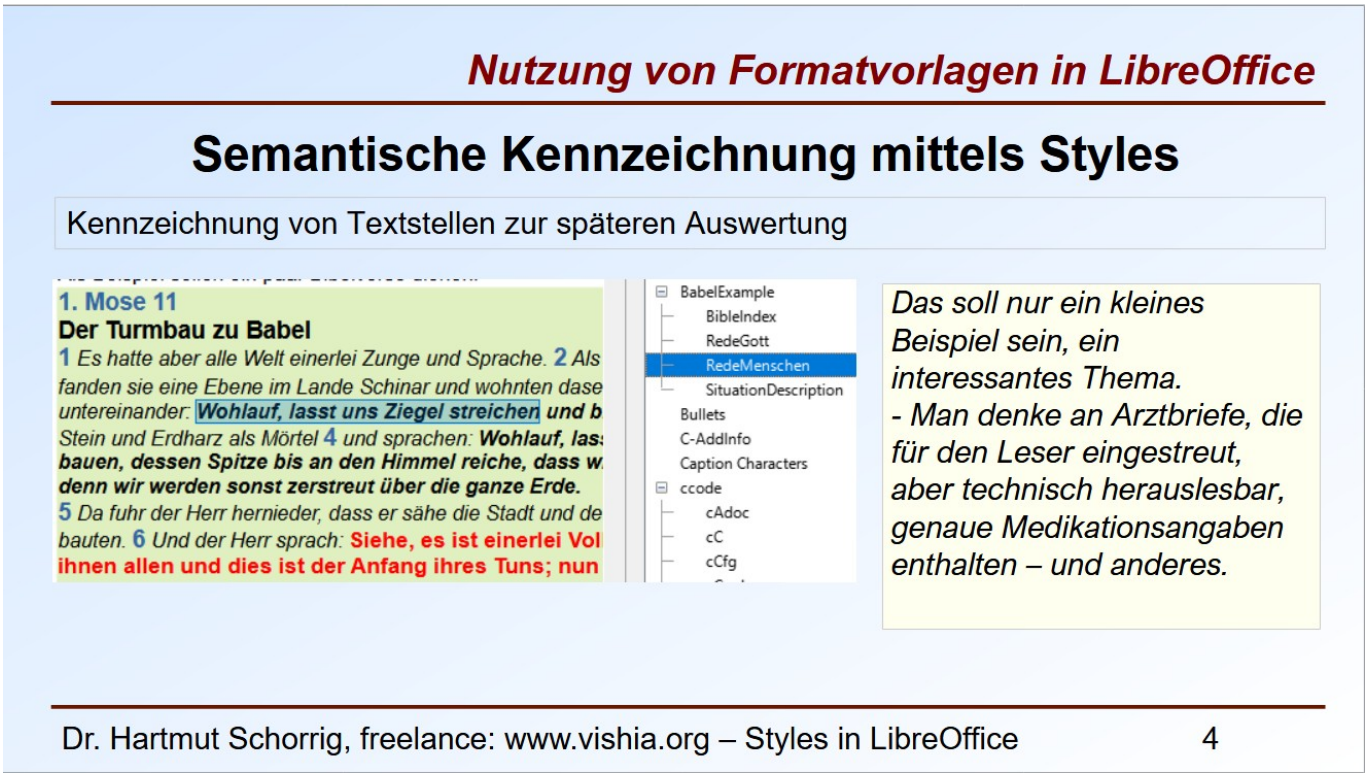
Die Bibel verwendet einheitliche Kennzeichnungen der Textpassagen. Diese sind hier optisch fett in blau etwas größer dargestellt und gut erkennbar. Für die Verarbeitung der Bibelstellen sind sie über den Style `BibleIndex` erkennbar. Damit sind die nachfolgenden Texte einorden- und auffindbar. Die Bibel-Indizes kennt jeder Pfarrer und auch fast jeder kirchlich orientierte Laie.

Die Überschrift ist mit dem Absatz-Style `BibleTitle` gekennzeichnet. Diese Titel sind Überschriften zu Verskapiteln. Ebenfalls sowohl optisch gut erkennbar als auch in Zuordnung des Textes automatisch herauslesbar.

Für das Verständnis des Textes oder auch einer Analyse ist der Text hier in drei Gruppen geteilt:

- Situationsbeschreibung, Character Style `SituationDescription`, hier kursiv schwarz.
- Rede von Menschen: Character Style `RedeMenschen`, hier kursiv fett schwarz.
- Aussagen Gottes: Character Style `RedeGott`, fett in rot.

Mit dieser Style-Kennzeichnung können nun beispielsweise alle Aussagen Gottes aus einem längeren Dokument herausgelöst werden, zusammen mit den davor stehendem `BibleIndex`, so aufbereitet kann beispielsweise linguistisch analysiert werden, wie Gott spricht bzw. es Martin Luther übersetzt hat. Desgleichen mit der Situationsbeschreibung und der Rede der Menschen in Luthers Übersetzungsstil.



S4.png

Ob sich dieses Schema auf die gesamte Bibel anwenden lässt, dazu gibt es hier keine Aussage. Es ist lediglich ein Beispiel, ohne wirklichen Hintergrund. Man denke aber an andere Beispiele, wie eine technische Beschreibung, die wirklich erklärt, wie etwas zusammenhängt. Aber in diesem Text sind genaue Angaben zu Anforderungen eingestreut, (*"Requirement Engineering"*), die dann herausgelöst als formaler Input für Tests dienen. Dieser Test braucht die Erklärung drumrum nicht. Aber es ist jederzeit möglich, wieder auf die Erklärung zu referenzieren. Selbstverständlich kann damit auch ein Content Management System gebaut

werden, also eine automatische (Um-) Generierung der Texte für bestimmte Art der Beschreibungen. Oder man denke an Arztbriefe, verständlich abgefasst, die mit Styles versehen bestimmte genaue Angabe für die Medikamenteneinnahme enthalten, als vielleicht interessanter Beitrag zur Digitalisierung im Gesundheitswesen ohne die von Einigen angestrebte zentrale Datenspeicherung. Ein solcher Arztbrief kann dann digital aber persönlich (auf einem Stick etwa oder tatsächlich auf der persönlichen Gesundheitskarte gespeichert) der Apotheke übergeben werden, die dann die notwendigen Informationen automatisch herausliest.

Nun kann man eben neben dem automatischen Herausziehen von den per Style gekennzeichneten Textteilen für eine Analyse auch die Erscheinungsform ändern. So ist hier die Aussage Gottes extra rot herausgehoben. Für andere Zwecke kann beispielsweise die Situationsbeschreibung herausgehoben dargestellt werden, in einem anderen Dokument, diese Textpassage original ohne Bearbeitung kopiert, in dem anderen Text nur die Erscheinungsform der Styles, der Indirekten Formatierung geändert. In einem Dokument lässt sich das nicht mischen. Allerdings wäre auch eine automatische Generierung des Dokumenteninhalts mit Varianten der Stylenamen möglich, um dies in einem Dokument zu zeigen.

5 Englische oder deutsche Bezeichnungen für die Indirekten Formatierungen

S5.png

Wie der geneigte Leser bemerkt hat, bin ich beim Schreiben bereits nicht vollständig auf deutsch orientiert, sondern rede von *Styles* anstatt von *Formatvorlagen*. Nunja. Wenn man bedenkt, dass um uns herum eine ganze Menge Menschen aus anderen Ländern ähnlich arbeiten und die babylonische Sprachverwirrung uns eben viele Sprachen beschert hat, so ist es ein Segen, dass Textinhalte heutzutage automatisiert ganz gut übersetzt werden können. Das trägt letztlich auch zur Völkerverständigung bei. Die Style-Bezeichnungen selbst sind aber technisch verwendet, im Verarbeitungsprogramm mit seiner Bezeichnung so hinterlegt, und nicht Bestandteil einer Übersetzung. Daher könnte es besser sein, hier eine einheitliche internationale Bezeichnungsweise zu verwenden. Früher wäre das in Europa Latein. Doch aktuell ist die Englische Sprache wohl der Standard, den die meisten Programmierer und auch

das einfache Volk mehr oder weniger verstehen. Daher verwende ich (in-)konsequent Englisch und rechne damit, dass ein englisch sprachkundiger Softwarebearbeiter mit der hier eher lax verwendeten Style-Bezeichnung wie `RedeGott` zurechtkommt. Es ist halt nur ein Identifier (Bezeichnung). Nebenbemerkung: CamelCase - Schreibweise ist durchaus geeignet (gemischt Groß/Kleinschreibung im Wort).

Im LibreOffice gibt es mit der Orientierung auf nationale Sprachen für die große Gruppe der nicht software/ingenieurtechnischen Anwender auch die Style-Namen in der Nationalsprache. Intern werden dennoch immer die einheitlichen Namen (in der Regel Englisch) gespeichert. Das ist wichtig für die automatisierte Verarbeitung, die nur einheitliche intern verwendete Style-Bezeichnungen berücksichtigen muss. Für selbst definierte Style-Bezeichnungen scheint diese Möglichkeit aber so nicht zu existieren. Sie ist mit Adaptionen sicherlich möglich, bedeutet aber zusätzlichen Pflegeaufwand.

Die Bezeichnungen der Styles lassen sich in einem LibreOffice Dokument ändern, ohne dass etwas an Text und Erscheinung geändert wird. Man kann also die Namen auch im Nachhinein nochmals ändern und anpassen. Für mehrere Texte artet das allerdings in Arbeit aus. Hier würde ein Tool helfen, für *„Renaming of style idents“*, dass ich möglicherweise demnächst noch schreiben und als Open Source verfügbar machen könnte.

Interessant ist es, dass es einige Bezeichnungen gibt, die im Englischen und im Deutschen ähnlich lauten, eventuell im Deutschen nicht die geläufigste aber doch verständliche Bezeichnung ist. Möglicherweise könnten solchen Bezeichnungen der Vorzug gegeben werden. Ich hätte mich also bemühen müssen, für „`RedeGott`“ eine Bezeichnung zu finden, die sowohl in Deutsch als auch in Englisch gängig oder verständlich ist. „`BibleIndex`“ erfüllt diese Voraussetzung.

6 Nur indirekte Formatierungen verwenden oder auch direkt formatieren?

Es kommt darauf an. In den Draw-Diagrammen für die Funktionsblock-Grafiken ist die direkte Formatierung, beispielsweise mit einer bestimmte Füllfarbe, definitiv zugelassen. Der Style-Bezeichnungen der Indirekten Formatierung ist für die semantische Kennzeichnung wichtig und muss entsprechend verwendet werden. Aber das Aussehen spielt für die Verarbeitung keine Rolle.

In diesem Dokument sind für den Text ausschließlich indirekte Formatierungen verwendet. Das liegt auch daran, dass der Text über eine Flachtext-Präsentation bearbeitbar ist, siehe https://vishia.org/LibreOffc/html/Videos_LOffcZmL.html. Bestimmte direkte Formatierungen werden in Indirekte umgewandelt, insbesondere „italic“ und „bold“ bzw. deutsch „kursiv“ und „fett“ in die Character-Styles *„Quotation“*, *„Emphasis“* und *„Strong Emphasis“* oder mit deutscher Spracheinstellung im LibreOffice *„Zitat“*, *„Betont“* und *„Stark betont“*. Diese Style-Namen drücken aus, was dargestellt wird (Quotation = ein Zitat) und nicht wie (kursiv). Sie sind entsprechend auch adaptierbar.

7 Markup Languages – Auszeichnungssprachen

S6.png

HTML = HyperText Markup Language ist in einem Forschungsprojekt von 1989 bis 1992 entwickelt worden, mit dem Ziel, umfangreiche und komplexe Texte dennoch mit geringem Datenaufwand über das neu geschaffene Internet austauschen zu können (CERN). Siehe https://de.wikipedia.org/wiki/Hypertext_Markup_Language. XML war die logische Weiterentwicklung des Konzeptes, ebenfalls unter dem Schirm des damals gegründeten W3C – *World Wide Web Consortium*. Texte werden mit semantischen Zusatzinformationen versehen und sind damit unabhänig (mit verschiedenen Geräten, Betriebssystemen) sowohl darstellbar als auch auswertbar.

Als XML etabliert war, sind fast alle Hersteller von verschiedensten Programmen dazu übergegangen, ihre eigene interne Darstellung, teils binär, teils spezifisch textuell, aufzugeben zugungsten der Speicherung in XML. Dies ist nicht ureigenstes Interesse der Hersteller, die oft eine Kundenbindung durch spezielle Formate erreichen wollen. Es ist der Druck der Nutzer für Austauschbarkeit und Offenheit. Es war dann einfach auch üblich geworden, XML-basierend zu speichern. Bei Mathworks-Simulink ist das ‚neue‘ XML-Format erst gegen etwa 2012 erschienen. Word für Windows hat sich gleich schon 2001 etwa angepasst, allerdings dann mit einigen verschiedenen Entwicklungsstufen des sogenannten „Office Open XML“ oder OOXML, siehe auch https://de.wikipedia.org/wiki/Office_Open_XML .

Dagegen war das Libre- oder auch Open-Office-Format ODF (Open Document Format) von OASIS = *„Organization for the Advancement of Structured Information Standards“*, siehe https://de.wikipedia.org/wiki/OpenDocument von Anfang an stark auf Offenheit und Standardisierung ausgelegt. Beide Formatnamen klingen ähnlich, bitte nicht verwechseln. Das ODF-Format wurde von Sun-Microsystems als damiliger Eigner von Star-Office 2006 als Norm ISO/IEC 26300 erstmals veröffentlicht und wird aktuell von OASIS gepflegt.

Schon in nicht-Computer-Zeiten war es üblich, den reinen Text für Druckerzeugnisse mit Schreibmaschine geschrieben weiterzugeben, mit entsprechenden Anmerkungen, wie er verwendet werden soll. Die genaue Auswahl der Schrifttypen (Bleisatz) oblag dann dem Hersteller in der Druckerei, der die entsprechenden Fachkenntnis in der Setztechnik hatte.

S7.png

Auf diesen Gedanken basieren letztlich auch solche Computer-Textbeschreibungssysteme wie Latex.

Bei Asciidoc ist im Namen bereits enthalten, dass dies ein „Flachtext-Format“ ist, also einfacher Text, mit jedem Texteditor bearbeitbar (ASCII-Zeichen, ASCII = *American Standard Code of Information Interchanging*, seit den 1960-ern weltweit übliche Zeichencodierung, heute mit dem aufwärtskompatiblen UTF-16, UTF-8 etc. erweitert).

ZmL ist meine eigene Entwicklung mit dem (gelungenem) Versuch, zwischen Flachtext und LibreOffice odt Writer zu konvertieren. Die Syntax von Asciidoc, die zuerst im Fokus stand, hat sich als nicht ausreichend erwiesen. Die Kapitelgliederung ist aber Asciidoc-Kompatibel. Beim Editieren mit einem Asciidoc-Editor hat man in der sogenannten Outline-Ansicht dann sofort die Kapitelgliederung im Blick.

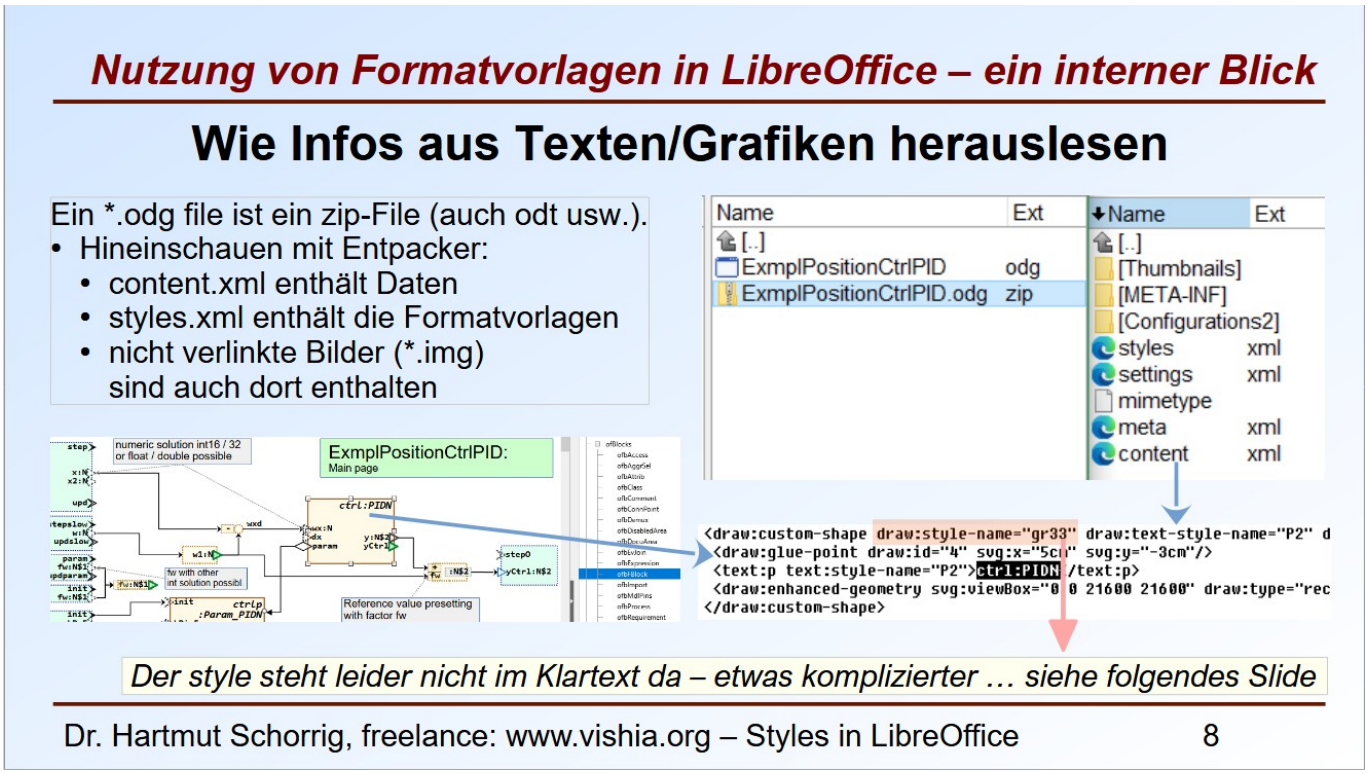
```
== Kapitelüberschrift <:#LabelKapitel.>

Flachtext des Kaptitels für Standard-Absatz-style

<:Code:Cpp>
  code_snippet(x, y);  // erzeugt dieses Absatzformat für code 'CodeCpp'
<.Code>
```

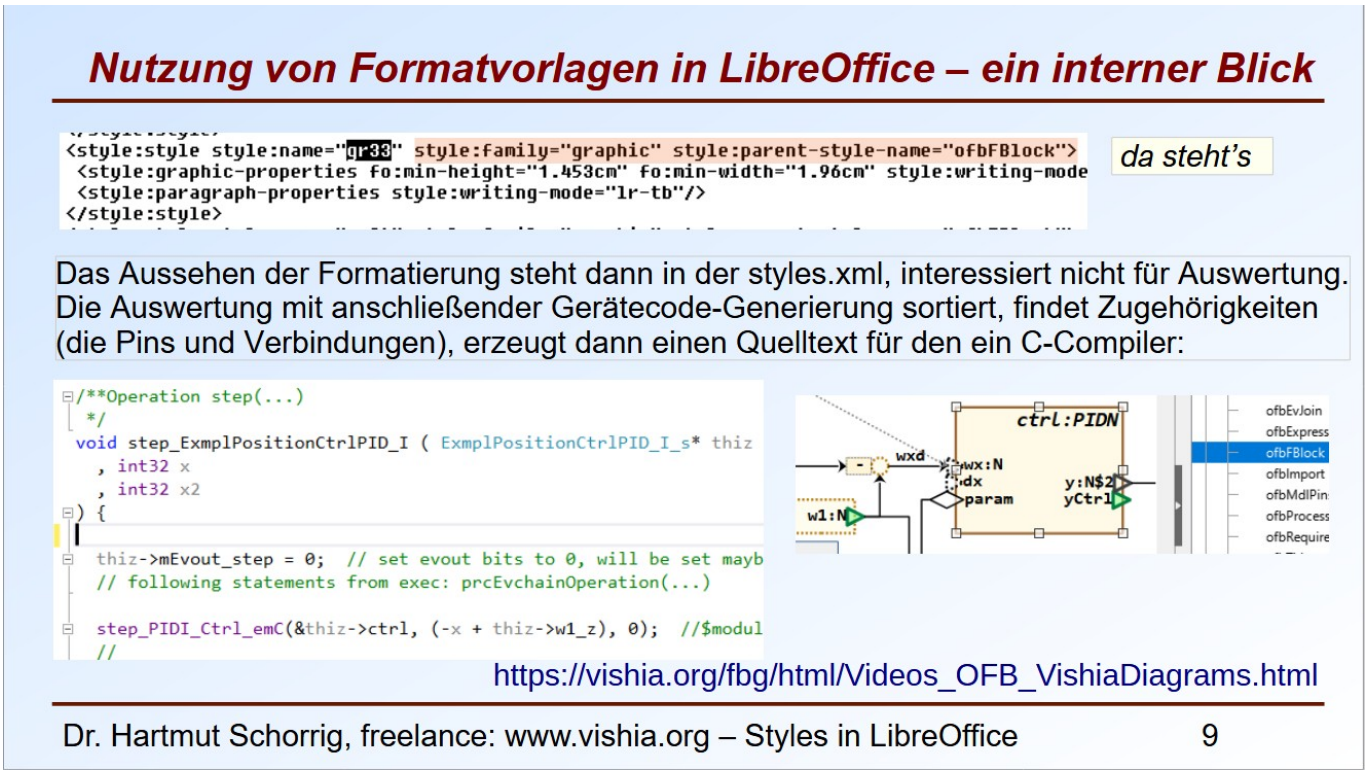
Siehe https://vishia.org/LibreOffc/html/Videos_LOffcZmL.html

8 Wie Infos aus Texten/Grafiken herauslesen



S8.png

Der dargestellt Funktionsblock `ctrl:PID` findet sich in den XML-Daten mit einem Style, zunächst als `draw-style-name="gr33"` gekennzeichnet ist, der dann aber



S9.png

in einem anderen Bereich des `content.xml` auf den `parent-style-name="ofbFBlock"` verweist.

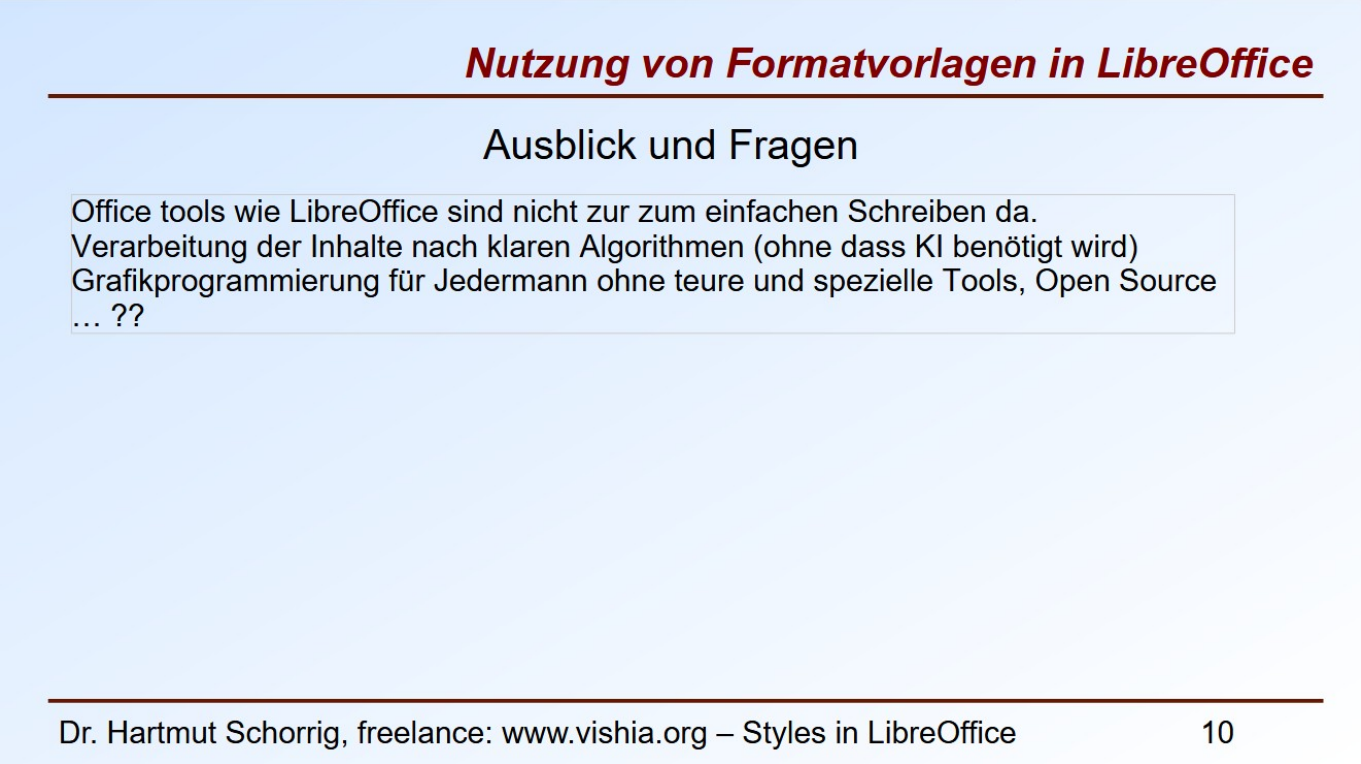
Wie im Bild dargestellt steckt in einer odg- oder odt-Datei XML drin. `content.xml` enthält den Inhalt. In meinem Beispiel für die Interpretation der Grafikdaten für Grafische Programmierung wird das XML von einem Java-Programm ausgelesen und konvertiert.

Die Speicherform von LibreOffice Text- oder Grafik-Dokumenten, das ODF = Open Document Format, ist letztlich auch eine Markup-Language. Eine wichtige Eigenschaft ist die Langzeitstabilität. Man ist nicht an bestimmte Versionen gekoppelt, es besteht meist auch eine genügend gute Auf- und Abwärtskompatibilität bei Standarderweiterungen.

Im Slide S9 ist der dann im Visual Studio (Integrierte Enwicklungsumgebung für C/++-Programmierung) der erzeugte C-Code geöffnet.

Dieses Tool kann über https://vishia.org/fbg/html/Videos_OFB_VishiaDiagrams.html angesehen und downgeloaded werden.

9 Ausblick und Fragen



S10.png