VSTENO Dokumentation

09.07.2019

Marcel Maci

Zusammenfassung

VSTENO - kurz Vector Shorthand Tool with Enhanced Notational Options oder zu Deutsch Vektor-Kurzschrift-Werkzeug mit erweiterten Darstellungsoptionen - ist ein PHP-Programm zur automatisierten Generierung von Kurzschrifttexten (Stenogrammen) unter Verwendung des SVG-Grafikformates. Die generierten Texte können als HTML-Seite in einem Browser angezeigt oder von dort als PDF exportiert und zum Beispiel auf einem E-Reader gelesen werden.

Den Kern des Programmes bildet die so genannte Steno Engine, ein abstraktes (verallgemeinertes) Stenografiezeichen-Satzsystem, das grundsätzlich an kein spezifisches Kurzschrift-System gebunden ist. Es können also eigene Zeichen und REGEX-Regeln zur Übertragung von Langschrift zu Kurzschrift programmiert werden. In der Grundversion kommt VSTENO mit Definitionen für das deutsche, spanische und französische System Stolze-Schrey (Grundschrift).

VSTENO ist Freie Software - das Programm kann unter der GPL (General Public License) frei kopiert und genutzt werden. Die aktuelle und hier dokumentierte Version ist 0.1rc (rc = release candidate oder Beta-/Testversion) und richtet sich sowohl an Anwender/innen, Linguist/innen und Programmierer/innen.

Einleitung

Die letzte Dokumentation von VSTENO liegt schon einige Zeit zurück[1]. Seitdem ist viel passiert: Datenbankfunktionen, Formelsprache (mit Branching und Stages), linguistisches Analyse-Modul (linguistical analyzer), Weiterentwicklung der Steno Engine (diakritische Zeichen und verbesserte Darstellung von Punktschlingen), spanisches und französisches System[2] - dies alles sind Neuerungen, die bis heute nirgendwo erläutert wurden.

Höchste Zeit also, dies nachzuholen und so Ihnen, den Anwender/innen die Möglichkeit zu geben, eigene Stenografie-Systeme umzusetzen. Denn dies ist zugegebenermassen eine der primären Visionen von VSTENO: Stenograf/innen anderer Systeme dazu zu animieren, ihr eigenes Stenografie-System mit VSTENO umzusetzen!

Wie jedes quelloffene, freie Software-Projekt strebt VSTENO danach, irgendwann jenes entscheidende Momentum zu gewinnen, das den Quantensprung vom Einpersonen- ins Community-Projekt ermöglicht. Mir ist sehr wohl bewusst, dass Stenografie heutzutage nicht mehr "so der Renner" ist, aber die Möglichkeit, dank VSTENO in Zukunft jeden beliebigen Roman in einem x-beliebigen Kurzschrift-System zu lesen, könnte vielleicht doch dazu führen, dass die Stenografie wieder etwas populärer wird.

Diesbezüglich kann ich aus eigener Erfahrung nur sagen, dass das Verwenden der Kurzschrift umso mehr Spass macht, je besser man sie beherrscht; und zur Beherrschung gehört nicht nur das Schreiben, sondern vor allem auch das (flüssige) Lesen. Gerade hier kann VSTENO mit seinem "Lesestoff à gogo" einen wesentlichen Beitrag leisten!

Sollte jemand Interesse haben, VSTENO weiterzuentwickeln - als Linguist/in (andere Systeme) oder als Programmierer/in (Erweiterung des Programmes), dann bitte bei mir melden (m.maci@gmx.ch)!

Nun wünsche ich viel Spass und anregende Lesestunden mit VSTENO!

Praktischer Teil

Dieser Teil ist für all jene gedacht, die VSTENO zur Generierung von "Lesestoff" verwenden wollen. Sie erfahren hier, wo Sie gemeinfreie Bücher finden, die Sie mit VSTENO übertragen können, wie Sie die Bücher in Kurzschrift übertragen und wie Sie die Stenografie-Texte ausdrucken oder auf einem E-Reader lesen können.

[1] Anwender/innen-Tutorial (https://www.vsteno.ch/docs/vsteno_tutorial.pdf) vom 11. August 2018 und Tutorial für Linguist/innen (https://www.vsteno.ch/docs/tutorial_linguistinnen.pdf) vom 18. August 2018. Die erste Version dieser Dokumenation wurde am 10.02.19 erstellt.

[2] Das spanische System wurde ab 3. Juni 2019 entwickelt und am 8. Juni erstmals auf www.vsteno.ch aufgeschaltet. Es wird fortan parallel zum deutschen System weiterentwickelt. Idem für das französische System, das Anfang Juli 2019 aufgeschaltet wurde.

Bücher

Jede/r Autor/in ist Urheber/in seiner/ihrer Bücher und hält somit sämtliche Rechte an seinen/ihren Texten. Deshalb die wichtigste Bemerkung vorweg: VSTENO will keinesfalls dazu aufrufen, urheberrechtlich geschützte Bücher ohne Abgeltung der entsprechenden Gebühren, die den Urheber/innen zustehen, zu verwenden!

Es gibt allerdings auch Bücher, die gemeinfrei sind, entweder weil deren Urheberrechte abgelaufen sind (dies ist in der Regel 70 Jahre nach dem Tod des/der Autor/in der Fall) oder weil sie von Anfang an als gemeinfrei (z.B. mithilfe von Creative Commons[3]) publiziert wurden. Eine Seite, die eine Sammlung von Büchern und Texten mit abgelaufenem Copyright anbietet, ist das Projekt Gutenberg:

http://gutenberg.spiegel.de/

Sie dürfen hier also jeden Text kopieren und für sich persönlich verwenden[4]. Sollten Sie die Bücher anderweitig verwenden wollen (also insbesondere öffentlich oder gar kommerziell), so klären Sie bitte vorgängig ab, ob und von welcher Art Copyright sie allenfalls betroffen sind!

Übertragen

Um Texte zu übertragen, öffnen Sie Ihren Webbrowser[5] und geben die Adresse www.vsteno.ch ein. Es existieren grundsätzlich zwei mögliche Formate, um Texte zu übertragen und zu lesen: (1) im Webbrowser und (2) auf einem E-Reader (via PDF).

Webbrowser

Um lediglich etwas Text im Webbrowser zu lesen, klicken Sie in der linken Navigationsleiste auf Mini. Sie können den Text entweder selber eintippen oder aus der Zwischenablage kopieren. Klicken Sie anschliessend auf abschicken - fertig!

Stenogramme, die Sie auf diese Weise generieren, werden als einzelne SVG-Grafiken (ein Bild pro Wort) inline in den HTML-Code eingefügt. Vorteil dieses Modus: Wenn Sie die Grösse des Browserfensters ändern, werden die Stenogramme automatisch neu angeordnet. Nachteil: Relativ grosser Abstand von Zeile zu Zeile (da jedes einzelne SVG-Bild die maximale Anzahl von 7 Stenografiehöhen enhalten muss, damit höher und tiefer gestellte Zeichen dargestellt werden können).

[3] https://de.wikipedia.org/wiki/Creative_Commons

[4] Mit "persönlich" ist eine nicht-kommerzielle, private Nutzung gemeint. Untersagt ist ausdrücklich eine kommerzielle, öffentliche Nutzung. Es gilt also zu unterscheiden, dass die ursprünglichen Texte zwar copyright-frei sind, das Projekt Gutenberg (bzw. deren Mitarbeitende) indes die Rechte der digitalisierten Version innehaben! Nur Bücher, welche ausdrücklich durch eine CC-Lizenz freigegeben sind, dürfen - unter der Berücksichtigung der angegebenen Rechte - öffentlich (und zum Teil auch kommerziell) weiterverwendet werden!

[5] Empfohlen wird Firefox. VSTENO ist Freie Software und wurde somit ausschliesslich unter Linux und unter Verwendung von Freier Software entwickelt. Getestet wurde VSTENO ausschliesslich auf den Browsern IceCat und ABrowser (unter Trisquel 7). Einige Funktionalitäten (insbesondere von VPAINT, welches JavaScript verwendet) laufen ausdrücklich nicht auf Browsern wie Safari und Internet Explorer.

E-Reader

Vorbemerkung

Um es gleich vorwegzunehmen: Kurzschrift-Texte auf einem E-Reader zu lesen ist einfach top! Nicht nur ist die Schriftqualität wesentlich besser als auf einem herkömmlichen Bildschirm oder Tablett, sondern Sie tun damit auch Ihren Augen etwas Gutes! E-Reader verwenden nämlich die so genannte E-Ink-Technologie (bei denen schwarze Punkte durch echte Pigmente dargestellt werden), die das Auge wesentlich weniger ermüden. Ich möchte deshalb die Anschaffung eines E-Readers allen empfehlen, die sich ernsthaft mit dem Gedanken tragen, längere Stenografie-Texte zu lesen. Auch in ökologischer Hinsicht ist ein E-Reader aus meiner Sicht eine gute Sache, da die Texte in guter Qualität gelesen werden können, ohne Papier verschwenden zu müssen.

Die Empfehlung für einen E-Ink-Reader kommt mir allerdings nicht leicht über die Lippen: Der Markt der E-Ink-Reader wird heute hauptsächlich dominiert von Amazon. Zu Amazon gibt es aus meiner Sicht nicht viel Gutes bzw. eigentlich nur Schlechtes zu sagen[6]. Das Dilemma an E-Readern ist im Moment, dass es wenige (bis keine) ethischen Alternativen gibt. Deshalb werde ich hier kein bestimmtes Modell nennen, sondern empfehle allen, sich ein eigenes Urteil zu bilden[7].

PDF

Es empfiehlt sich Kurzschrift-Texte via PDF auf den E-Reader zu transferieren. Dies kann in zwei Schritten erreicht werden: (1) Generieren der gelayouteten Seiten im Webbrowser und (2) Export der Seiten als PDF (via Druckfunktion des Browsers).

1. Gehen Sie auf die Seite www.vsteno.ch und wählen Sie Maxi aus der linken Navigationsleiste. Geben Sie den Text ein (oder kopieren Sie ihn aus der Zwischenablage) und wählen Sie dann folgende Einstellungen: Fenster = Vollseite (ohne Button); Ausgabe = Layout (statt Inline). Ebenfalls empfiehlt sich unter Header, Titel und Einleitung entweder abzuwählen (dann wird gleich mit dem Kurzschrift-Text begonnen) oder hier einen eigenen Titel und eine Einleitung einzutragen (dies erscheinen dann in Langschrift zu Beginn des Textes). Die übrigen Einstellungen können Sie vorläufig belassen. Klicken Sie anschliessend auf abschicken.

2. Im Unterschied zu Inline erscheint ihr Text nun gelayoutet, d.h. für jede Seite wird nun eine grosse SVG-Grafik erstellt und die Stenogramme darin platziert (standardmässig ist die Seite 660x1000 Punkte gross und die Stenogramme

[6] Ich verweise hier nurmal auf Richard Stallmans Seite: https://stallman.org/amazon.html. Aus persönlicher Sicht füge ich hinzu, dass ich es stossend finde, dass ein Mensch wie Jeff Bezos, seines Zeichens reichster Mann der Welt, ein weltweites Unternehmen führt, das Angestellte unter unmenschlichen Arbeitsbedigungen ausbeutet, im Namen der Globalisierung jeden lokalen Markt kaputt macht - und schliesslich noch die Frechheit hat, Steuerschlupflöcher zu nutzen, die dazu führen, dass die Firma Amazon im Jahr 2018 nicht nur keine Steuern bezahlt hat, sondern sogar noch 129 Millionen Dollar vom Fiskus rückerstattet erhielt (siehe hier: https://www.nau.ch/news/wirtschaft/amazon-zahlt-in-den-usa-keine-steuern-65485543).

[7] Hilfreich dabei ist aus meiner Sicht die Seite: https://www.ethicalconsumer.org/technology/shopping-guide/tablets-e-readers

werden im Blocksatz dargestellt). Diese Seiten können Sie nun zu einem PDF exportieren, indem Sie die Druck-Funktion Ihres Browsers nutzen. Dies kann je nach Browser etwas unterschiedlich sein (meistens muss im Menü Datei der Punkt Drucken gewählt werden), wichtig ist, dass Sie hier die Option "Druckvorschau" oder "Drucken in Datei" wählen. Der Trick besteht also darin, die Druckausgabe nicht auf den Drucker zu schicken, sondern in eine Datei umzuleiten.

Übertragen Sie anschliessend das PDF auf Ihren E-Reader.

Längere Texte

Texte bis zu einer Länge von etwa 20 Stenografieseiten können im Moment mit VSTENO problemlos verarbeitet werden[8]. Bei längeren Texten tritt der Fehler "Gateway timeout" auf, da die Generierung zu lange dauert. Als Faustregel: VSTENO benötigt etwa 20 Sekunden für das Generieren einer Stenografie-Seite. Der Fehler "Gateway timeout" tritt somit in etwa nach 7 Minuten auf.

Der grösste Teil der Rechenzeit, nämlich etwa 80%, entfällt hierbei nicht auf das Generieren der Stenografie-Zeichen als solche, sondern auf die vorgeschaltete, linguistische Analyse. Damit trotzdem auch längere Texte (wie z.B. Romane) mit VSTENO bearbeitet werden können, empfiehlt es sich, die linguistische Analyse abzukoppeln bzw. die Generierung zu Portionieren. Zwei Vorgehen bieten sich hierfür an:

1. Verwenden der LNG-Form: Kopieren Sie den Langschrift-Text ins Textfeld des Maxi-Formulars, wählen Sie anschliessend unter Ausgabe die LNG-Form (zusätzlich empfiehlt es sich auch, die Optionen "Vollseite" und "ohne Button" zu markieren) und anschliessend "abschicken". VSTENO wendet nun nur die linguistische Analyse an und gibt das Resultat im Browser aus. Achten Sie auch hier darauf, dass die Berechnung des eingegebenen Textes ungefähr 7 Minuten Rechnzeit nicht überschreitet. Wiederholen Sie anschliessend die Berechnung mit weiteren Textteilen und kopieren Sie das Resultat fortlaufend in eine ASCII-Textdatei (hierfür kann ein ganz normaler Texteditor verwendet werden). Kopieren Sie am Schluss den kompletten Text in LNG-Form ins Textfeld des Maxi-Formulars, wählen Sie unter "Text" das Format "Meta (LNG)" und als Ausgabe-Format "Layout" (mit entsprechenden Werten) und generieren Sie anschliessend das komplette PDF. Da VSTENO nun die linguistische Analyse nicht mehr anwenden muss, läuft die Generierung des Stenografie-Dokuments sehr schnell ab (wenige Minuten für 150-200 Seiten).

2. Generieren separater PDFs: Falls der Ursprungstext in Kapitel unterteilt ist, die eine Länge von 20 Seiten nicht überschreiten, können Sie auch jedes Kapitel einzeln als PDF generieren und anschliessend mit dem nachfolgenden Kommandozeilen-Befehl zusammenfügen[9]. Beachten Sie, dass VSTENO hier

[8] Bezieht sich auf www.vsteno.ch und die jetzige Server-Konfiguration

[9] Dies funktioniert unter Linux mit dem Programm GhostScript (gs). Andere Betriebssystem bieten vermutlich ähnliche Hilfsprogramme, mit denen PDFs zusammengeführt werden können.

in jedem PDF mit der Seitennummerierung neu beginnt (falls Sie also eine fortlaufende Seitennummerierung möchten, so müssen Sie diese manuell für jedes PDF unter "Layout" anpassen):

```
gs -q -sPAPERSIZE=a4 -dNOPAUSE -dBATCH -sDEVICE=pdfwrite
-sOutputFile=pdf_komplett.pdf pdf_teil1.pdf pdf_teil2.pdf
```

Tipps

Zeilenumbrüche

Sie werden vielleicht bereits bemerkt haben, dass VSTENO normalen Text als fortlaufend betrachtet (es nützt also nichts, wenn Sie im Textfenster zusätzliche Zeilenumbrüche und Leerzeilen einfügt - VSTENO wird diese einfach ignorieren). Um einen Zeileneinbruch einzufügen müssen Sie die HTML-Tags
 (break) und (paragraph) verwenden[10]:

> Dies ist die
> erste Zeile
 dies ist
> die
> zweite Zeile

Dieses Beispiel wird also in zwei Zeilen ausgegeben.

Damit Sie - insbesondere bei längeren Texten (wie Büchern), die Sie von einer Webseite via Zwischenablage kopieren - nicht alle Formatierungen von Hand neu eingeben müssen, empfiehlt sich folgendes Vorgehen:

- Rufen Sie die Seite mit dem Text auf, den Sie kopieren möchten[11].
- Wählen Sie "Seitenquelltext anzeigen" o.ä. aus Ihrem Browsermenu.
- Scrollen Sie zu der Stelle, die den Text enthält.
- Wählen Sie mit der Maus den gesamten Text (inklusive HTML-Tags) aus, den Sie kopieren möchten.
- Fügen Sie diesen Text (inklusive HTML-Tags) ins Textfeld von VSTENO ein und berechnen Sie die Seite.

Hier unser Beispieltext als HTML-Quellcode:

[10] Neu bietet VSTENO die Option "Breaks" gleich oberhalb des Textfeldes: Ist "Breaks" gesetzt (Standard) übernimmt VSTENO die eingegebenen Zeilenumbrüche, FALLS der Text keine HTML-Tags enthält (sonst - oder wenn "Breaks" nicht markiert ist - werden Zeilenumbrüche wie beschrieben ignoriert).

[11] Wir verwenden hier im Folgenden die "Notwendige Vorrede" aus Matto regiert von Friedricht Glauser in der Version des Projekts Gutenberg, siehe http://gutenberg.spiegel.de/buch/matto-regiert-1855/1.

```
<h3>Notwendige Vorrede</h3> <p>Eine Geschichte zu erz&auml;hlen,
die in Berlin, London, Paris oder Neuyork spielt, ist ungef&auml;
hrlich. Eine Geschichte zu erz&auml;hlen, die in einer Schweizer
Stadt spielt, ist hingegen gef&auml;hrlich. Es ist mir passiert,
da&szlig; der Fu&szlig;ballklub Winterthur sich gegen eine meiner
Erz&auml;hlungen verwahrt hat, weil darin ein Back vorkam.
Ich mu&szlig;te dann den Boys und anderen Fellows best&auml;tigen,
da&szlig; sie nicht gemeint waren.</p>
```

Wie man sieht werden hier nicht nur alle Zeilenumbrüche wie
, mitkopiert, sondern auch Sonderzeichen wie ä (ä) oder das deutsche sz (ß) anders dargestellt. All das braucht Sie nicht zu kümmern: VSTENO wandelt diese Sonderzeichen um und verwendet die Tags, um den Text in Abschnitte zu unterteilen (wie es im Original der Fall war).

Einstellungen

Unter Zeichen haben Sie die Möglichkeit Grösse, Dicke, Neigung, Schattierung etc. anzugeben. Ein Hinweis gleich vorweg: Nicht alle Optionen sind bereits in beiden Formaten (Inline und Layouted) implementiert! Am wichtigsten für ein gutes Schriftbild auf einem E-Reader (oder beim Ausdrucken) ist jedoch die Schriftgrösse und die Schriftdicke. Experimentieren Sie etwas mit diesen Werten, um die optimalen Grössen für Ihren E-Reader zu finden! Gleiches gilt für die Parameter unter Layouted: Experimentieren Sie mit Höhe/Breite, Randeinstellungen etc. um die für Sie optimalen Werte zu finden!

Inline-Option-Tags

Inline-Option-Tags bieten Ihnen eine weitere Möglichkeit, die Gestaltung eines Stenografie-Textes anzupassen. Inline-Option-Tags sehen HTML-Tags sehr ähnlich und können wie diese an jeder beliebigen Textstelle eingefügt werden.

```
<p>Dies ist ein
<@token_color=red>
rotes
<@token_color=black>
Wort.</p>
```

Beachten Sie, dass VSTENO den Text "Dies ist ein rotes Wort" auch hier als fortlaufende Stenogramme in einer einzigen Zeil ausgibt (Zeilenumbrüche haben keine Bedeutung[12]). Mit dem Inline-Option-Tag <@token color=red > wird die Zeichenfarbe nach "dies ist ein" auf rot gesetzt und das Wort "rotes" somit in roter Farbe

[12] Dies gilt, solange sie HTML-Tags, wie hier z.B. und , verwenden. Wenn Sie und weglassen, fügt VSTENO automatisch Zeilenumbrüche ein. Deaktivieren Sie in diesem Fall die Option "breaks" oben im Formular, um einen fortlaufenden Text zu erhalten.

ausgegeben. Analog wechselt <@token_color=black> die Schriftfarbe wieder zurück auf schwarz.

Inline-Option-Tags weisen somit folgendes Format auf:

<@variable="Wert">

<@variable='neuer Wert' >

<@variable=Wert >

Die Variable bezeichnet die Option, die angepasst wird, indem sie den Wert "Wert" zugewiesen erhält. Der Wert kann in doppelten oder einfachen Anführungszeichen oder auch direkt zwischen = und > stehen (wählen Sie hier die Form, die Ihnen persönlich am besten gefällt).

Variablen

Grundsätzlich können sämtliche Optionen der Steno-Engine - insbesondere also auch jene, welche in der Vollversion per Webformular zur Verfügung stehen - mit Inline-Option-Tags dynamisch, d.h. innerhalb des Textes, verändert werden. Die wichtigsten Variablen sind:

- token_size: Grösse der Stenozeichen (Zoomfaktor, Standard: Standard 1.6).
- token_type: Kann die Werte "shorthand" (Standard), "handwriting", "html-text" und "svgtext" annehmen. Dadurch kann kann Text in Langschrift und Stenografie gemischt werden (mehr dazu im Abschnitt "gemischte Texte").
- token_thickness: Liniendicke der Stenozeichen (Standard: 1.25).
- token_inclination: Neigung der Stenozeichen (Standard: 60 Grad).
- token_shadow: Stärke der Schattierung (Standard: 1.0).
- token_distance_wide: Abstand zwischen zwei weiten Stenozeichen (Standard: 15)[13].
- token_color: Farbe der Stenozeichen (Standard: "black" bzw. rgb(0,0,0)[14]).
- token_style_type: Linientyp mit dem Wert "solid" (durchgehend), "dotted" (gepunktet) oder "dashed" (gestrichelt).
- token_style_custom_value: benutzerdefinierter Wert für den Linientyp[15].

[13] Der weite Abstand entspricht im deutschen System den Vokalen e, ä (horizontal); ei, eu (1/2 Stufe höher); ü, ö (1/2 Stufe tiefer). Die enge Verbindung token_distance_narrow kann zwar ebenfalls eingestellt werden. Da Stenozeichen jedoch sehr individuelle Abstände benötigen, sollte hierfür stattdessen mithilfe des Spacers und des RX-GEN entsprechende Regeln definiert werden (siehe hierzu das entsprechende Kapitel).

[14] Hier kann aus HTML entweder die rgb-Notierung rgb(r,g,b) oder die vordefinierten Standarfarben (wie black, green, white, red, blue, purple etc.) verwendet werden.

[15] Hier können zwei durch Komma abgetrennte Werte zugewiesen werden: <@token_style_custom_value=1,1> bedeutet zum Beispiel: 1 Punkt zeichnen, einen Punkt leerlassen etc. (was einer eng punktierten Linie entspricht).

- svgtext_size: Grösse der Zeichen in Normalschrift in px (Standard: 30, es wird die Schrift Courier verwendet).

Sämtliche von VSTENO verwendeten Variablen sind in der Datei session.php aufgeführt. Nicht alle können durch den Nutzer geändert werden und bei anderen macht es wenig Sinn, sie im Laufe einer Berechnung zu verändern[16].

Gemischte Texte

Mithilfe der Inline-Option-Tags bietet VSTENO auch die Möglichkeit, Stenogramme und Langschrift zu mischen:

```
<p>Einige Kürzungen:
<@token_type=svgtext>
gegen
<@token_type=shorthand>
gegen
<@token_type=svgtext>
mit
<@token_type=shorthand>
mit</p>
```

Hier wird also nach den Wörtern "gegen" und "mit" in Langschrift die entsprechende Abkürzung als Stenogramm angezeigt.

Die Variable "token_type" kann folgende Werte annehmen:

- shorthand: Kurzschrift (es wird das vorselektierte Modell verwendet).
- handwriting: Langschrift handgeschrieben (Blockschrift im ähnlichen Stil wie die Stenozeichen)[17].
- htmltext: Langschrift als HTML-Text 9[18].
- svgtext: Langschrift als SVG-Grafik dargestellt.

Wie bereits angedeutet sind gewisse Optionen (darunter auch token_type) im Layout-Modus nicht oder nur eingeschränkt verfügbar. Zuverlässig verwendet werden können aber in beiden Modi: Schriftdicke (token_thickness), Schattierungsstärke (token_shadow), Farbe (token_color).

[16] Die Variable actual_model zum Beispiel kann nicht verändert werden, da VSTENO gleichzeitig immer nur 1 Kurschriftsystem verwenden kann. Die Variable token_size hingegen kann zwar angepasst werden, dennoch sollte dies mit Vorsicht getan werden: Wir die grösse der Stenozeichen während der Berechnung verändert, dann sollte dies nicht innerhalb der gleichen Zeile geschehen (da sonst die Grundlinie der Zeichen nicht mehr auf gleicher Höhe liegt) und im Layout-Modus sollte dies gar nicht verwendet werden (da der Layout-Modus mit fixen Zeilenabständen arbeitet, die dann nicht mehr zur Zeichengrösse passen). Variablen, die vom/von der Nutzer/in verändert werden können, sind in der Datei options.php in der Variable $whitelist aufgefüht.

[17] Dies ist im Moment noch nicht umgesetzt: Erst Zahlen und der Grossbuchstabe A können verwendet werden.

[18] Diese Variante bietet nur sehr eingeschränkte Möglichkeiten und wir in zukünftigen Versionen vermutlich wieder verschwinden.

Sprache

Im Abschnitt "Engine" kann die Sprache gewählt werden. Im Moment stehen Deutsch (standard[19]) und Spanisch (als weitere Option[20]).

Wenn Sie als Benutzer/in eingeloggt sind, wird hier zusätzlich ihr eigenes (editierbares) Modell angezeigt und diese Auswahl selektiert auch automatisch das aktive Modell, mit dem Sie arbeiten.

Zeichen \ und |

Diese Zeichen geben VSTENO an, ob es sich um ein zusammengesetztes Wort handelt und wie dieses geschrieben werden soll. Betrachten wird die beiden Wörter Lebenspartner und Eulenspiegel. Beide Wörter sind aus zwei Einzelwörtern zusammengesetzt und beide weisen an der Verbindungsstelle die Konsontengruppe -nsp- auf.

Im System Stolze-Schrey können diese auf zwei Arten gruppiert werden: (1) -ns- plus -p- oder (2) -n- plus -sp-. VSTENO hat grundsätzlich keine Möglichkeit zu entscheiden, welches die bessere Gruppierung ist. Sie können aber eine Gruppierung erzwingen, indem Sie die Einzelwörter mit | und \ trennen:

```
Lebens|partner, Lebens\partner[21]
Eulen|spiegel, Eulen\spiegel
```

Nun werden die Konsonantengruppen richtig gruppiert. Mit | bleiben die Wörter zusammegschrieben (fortlaufendes Stenogramm), mit \ hingegen kehrt VSTENO für das zweite Wort auf die Grundlinie zurück (die Wortteile werden getrennt, nahe nebeneinander geschrieben).

Drucken

Nebst der Ausgabe der Stenogramme im Webbrowser oder auf einem E-Reader können Sie diese natürlich auch ausdrucken. Auch hier wird (wie beim E-Reader) der Modus Layout (anstelle von Inline) empfohlen, da VSTENO dann ganze Seiten generiert (und die Stenogramme innerhalb der Seite ästhetisch - z.B. unter Verwendung von Blocksatz - anordnet). Ebenfalls wird empohlen, die Optionen Vollseite und ohne Button zu verwenden, damit VSTENO die Stenogramme auf eine leere Seite setzt (und nicht das Layout der Homepage www.vsteno.ch mitgedruckt werden muss).

Zum Drucken können Sie entweder ein PDF exportieren (wie für den E-Reader) oder die Druckfunktion des Browsers direkt verwenden. Auch hier empfiehlt es sich, verschiedene Einstellungen (Grösse, Dicke, Layout: Breite/Höhe, Rand etc.) auszuprobieren, um herauszufinden, welche Werte auf Ihrem Drucker die besten Resultate ergeben.

[19] Die "Glückskekse" (fortune cookies) rechts oben werden immer auf Deutsch angezeigt.

[20] Das Spanische System funktioniert erst rudimentär, es wird sich in den laufenden Monaten weiter verbessern

[21] Verwenden Sie zur Eingabe von \ und | die Taste AltGr + <> und AltGr + 7.

Versionen

Entwicklung

Um den Überblick über die aktuelle Version und die dokumentierten (bzw. nicht dokumentierten) Funktionen zu gewährleisten, hier ein kurzer, historischer Abriss über die Entwicklung von VSTENO:

VSTENO startete im April 2018 prinzipiell als Proof of Concept. Es sollte also erst einmal überprüft werden, wie und ob ein Stenografie-Satzsystem überhaupt umsetzbar wäre. In dieser Phase wurden grundsätzliche Konzepte festgelegt: Verwendung des Grafikformates SVG, Modellierung der Stenografiezeichen als Bezier-Kurven (Splines), Implementierung in PHP und Benutzung des Programmes via Webbrowser, spätere Integrierung einer Datenbank.

Im Mai 2018 gelangte eine erste Version auf Github[22]. Ziel fortan: Das Programm unter der GPL publik zu machen und sauber zu versionieren.

Im August 2018 war das Programm so weit fortgeschritten, dass eine erste offizielle Dokumentation geschrieben wurde[23]. Bereits damals konnten im Prinzip eigene Stenografie-Systeme umgesetzt werden, allerdings mussten die Daten direkt als Variablen in den PHP-Code integriert werden.

Bis zu diesem Zeitpunkt (August) konnte VSTENO nur regelmässige Stenogramme generieren, d.h. Wörter, die automatisiert von der Langschrift in die Kurzschrift übertragen werden. Ausnahmen - also Wörter, die von den Regeln abweichen - konnten nicht oder nur sehr rudimentär (mithilfe eines Trickster[24]) berücksichtigt werden. Aus diesem Grund erhielt VSTENO im September 2018 eine Datenbankanbindung. Abweichende Wörter konnten fortan in einem Wörterbuch hinterlegt werden.

Im November schliesslich konnte sich VSTENO endlich vom PHP-Code lösen: Eigene Stenografie-Systeme konnten nun mit einer eigenen Formelsprache definiert und als externe ASCII-Datei abgespeichert werden. Die Integration eines Parsers stellte einen wesentlichen Schritt in der Entwicklung einer abstrakten Steno Engine dar und legte den Grundstein für die Version 0.1.

Im Anschluss wäre es eigentlich möglich gewesen, diese Steno Engine weiterzuentwickeln, allerdings hatte sich im Laufe der Zeit gezeigt, dass der gewählte Ansatz auch Mängel aufwies: Schattierungen der Stenogramme waren nur durch abrupte Übergänge realisierbar (kein Anti-Aliasing, Treppenbildung), ebenso wurden gewisse Stenozeichen durch den Neigungsalgorithmus verformt. Es entstand deshalb die Idee, eine neue Steno Engine zu entwickeln: die Steno Engine 1 (SE1) sollte nur noch Bugfixes erhalten und später durch eine bessere Steno Engine 2 (SE2) ersetzt

[22] https://github.com/marcelmaci/vsteno

[23] Anwender/innen-Tutorial (https://www.vsteno.ch/docs/vsteno_tutorial.pdf) vom 11. August 2018 und Tutorial für Linguist/innen (https://www.vsteno.ch/docs/tutorial_linguistinnen.pdf) vom 18. August 2018.

[24] Der Trickster enthielt spezielle Regeln für unregelmässige Wörter. Regeln für Unregelmässiges - das scheint schon vom Prinzip her widersinnig. Kein Wunder also, dass der Trickster ein problematisches Konzept war und in VSTENO während geraumer Zeit sein (un)regelrechtes Unwesen trieb ...

werden.

Man sagt gemeinhin, dass man in der Regel mit 20% Aufwand 80% Erfolg erreicht. Und genau dies traf auf den geplanten Sprung von der SE1 zur SE2 zu. Während dreier Monate (von November bis Januar) schrieb ich im Hinlick auf die SE2 zuerst an einem grafischen Zeichen-Editor namens VPAINT in JavaScript[25]. Und je ausgefeilter VPAINT wurde, umso mehr begann die SE2 zu wuchern: Im Januar machte der Quellcode von VPAINT bereits die halbe Grösse des ursprünglichen PHP-Codes von VSTENO aus. Dabei war dies erst der Editor und die Implementierung der SE2 in PHP noch in weiter Ferne ...

Dies wurde insofern zu einem Dilemma, als dass jede linguistische Weiterentwicklung des Systems Stolze-Schrey blockiert war: Wenn die SE1 entfernt werden sollte, machte es keinen Sinn, die bereits vorhandenen Zeichen und Regeln weiterzuentwickeln (da sie für die SE2 nicht mehr gültig sein würden); andererseits konnten keine neuen Zeichen und Regeln definiert werden, solange die SE2 nicht fertig war.

Im Januar hielt ich es schliesslich für eine gute Idee, dem Dilemma durch die magischen Worte Backports und Rückwärtskompabilität zu entkommen: Es sollten also die wichtigsten Features der SE2 auf die SE1 rückportiert werden (damit insbesondere auch der Editor VPAINT für die SE1 verwendet werden könnte) und die neue SE1 (nun mit dem Namen SE1 rev1) sollte vollständig rückwärtskompatibel (zur ursprünglichen SE1 rev0) sein. Die SE1 rev1 sollte also das "Beste aus zwei Welten" vereinen - die aufgebretzelte eierlegende Wollmichsau, gewissermassen.

Aber schön ist bekanntlich (nur) die Theorie! Im Februar wurde klar, dass die Rückwärtskompatibilität sehr wackelig war und die zum Teil gewagten Backports die SE1 aus entwicklungstechnischer Sicht auf ein Alpha-Stadium zurückwarfen. Was nichts anderes bedeutete als: Mehr Zeit für Bugfixes zu investieren. Was nichts anderes bedeutete als: Wieder keine Zeit, um die Zeichen und Regeln für Stolze-Schrey weiterzuentwickeln. So traf ich Mitte Februar zwei Entscheidungen: (1) die SE2 und die SE1 rev1 wurden mit sofortiger Wirkung komplett deaktiviert und die Entwicklung radikal und für unbestimmte Zeit auf Eis gelegt, d.h. und bis zur offiziellen Release der Version 0.1 soll nur noch - und ausschliesslich - die SE1 rev0 weiterentwickelt werden; (2) die SE1 rev0 wird nicht (auch in Zukunft nicht) aus VSTENO verschwinden: Dazu ist sie trotz ihrer Nachteile zu gut und zu "rock solid"!

Der aktuelle Stand der Dinge ist somit: In diesem Dokument werden ausschliesslich Funktionen der SE1 rev0 erläutert. Das Programm VPAINT wird nur als Hilfsinstrument zur visuellen und interaktiven Darstellung der Stenozeichen erläutert. Von der Verwendung von VPAINT zur Speicherung von Zeichen und der Nutzung von Funktionalitäten, die zur SE2 oder zur SE1 rev1 gehören,[26] wird dringend abgeraten.

[25] VSTENO und etwaige Hilfsprogramme sollten unbedingt via Webbrowser benutzbar sein.

[26] Sofern sie nicht deaktiviert sind. Die meisten Funktionen der SE2 und der SE1 rev1 sind deaktiviert. Dennoch ist die Speicher-Funktion von VPAINT auch in der Version 0.1rc aktiv (da der Backport der SE1 rev1 indirekt funktioniert, d.h. die Zeichendefinitionen werden von VPAINT in den Text-Editor der rev0 exportiert, von wo aus sie von der ursprünglichen SE1 rev0 abgespeichert werden, als handelte es sich um Daten der SE1 rev0 - da VSTENO in der Version 0.1rc nur Daten der SE1 rev0 versteht, wird davon DRINGEND abgeraten: Inkompatibiltäten und korrupte Zeichendefinitionen sind garantiert).

Steno Engines

Wie bereits erläutert soll in dieser Dokumentation grundsätzlich nur auf die SE1 rev0 eingegangen werden. Dennoch hier ganz kurz die wesentlichen Unterschiede zwischen den Versionen[27]:

- Ursprüngliche Steno Engine 1 (SE1 rev0): Hier wird die Mittellinie der Zeichen anhand von aufeinanderfolgenden Bezier-Kurven (so genannte Splines) modelliert. Dickere und dünnere Stellen werden durch dickere und dünnere Striche dargestellt. Der Übergang von einer Strichdicke zur nächsten erfolgt abrupt (diskontiuierlich), was je nachdem eine Treppenbildung zur Folge hat. Die SE1 neigt Zeichen ausschliesslich durch horizontale Punktverschiebung, was zu einer Änderung der Winkel führt, sodass die Zeichen - je nach Art und Beschaffenheit - verformt werden. Die einzige Möglichkeit, mit der SE1 rev0 schöne Zeichen zu modellieren, besteht darin, das Zeichen direkt in der vorgesehenen Neigung zu entwerfen (VSTENO verwendet im Fall der Grundschrift Stolze-Schrey 60 Grad).

- Steno Engine Revision 1 (SE1 rev1): Verwendet die gleiche Mittellinienmodellierung und horizontale Punktverschiedbung wie die SE1 rev0. Zwei Features der SE2 wurden jedoch als Hacks rückportiert: (1) parallele Rotationsachsen, (2) orthogonale und proportionale Punktdrehung. Bei der Rückportierung reden wir ganz klar von einem Hack: Die SE1 war nicht dafür ausgelegt, solche Funktionalitäten zu integrieren! Beispielsweise werden in der SE1 rev0 nicht einzelne Zeichen geneigt, sondern es werden zunächst alle benötigten Zeichen aneinandergefügt und danach das ganze Wort geneigt (unter Verwendung der horizontalen Punktverschiebung). Die einzige Möglichkeiten, die Funktionen (1) und (2) zu integrieren, besteht deshalb darin, (a) die Zeichen einzeln zu neigen (wobei die SE1 rev0 immer noch glaubt, es seien nicht geneigte Zeichen) und (b) die Zeichen dann zusammenzufügen und nicht mehr zu neigen (die SE1 rev0 wendet dann eine Neigung um 0 Grad an). Die Umsetzung von (a) und (b) impliziert weitere Komplikationen: Koordinaten für parallele Rotationsachsen und für neue Punkttypen müssen im starren Datengerüst der SE1 rev0 unergebracht werden. Da dort praktisch kein freier Platz mehr besteht, müssen die zusätzlichen Informationen zum Teil bitweise mit bestehenden Elementen verknüpft (und nachher wieder ausgelesen werden). All diese Komplikationen führen dazu, dass die SE1 rev1 bis jetzt nur unzuverlässig funktioniert und deshalb in der Version 0.1rc deaktiviert wurde.

- Steno Engine 2: Hier wird ausgehend von einer Mittellinie (wie in der SE1) ein Umriss des Zeichens modelliert. Jedes Zeichen ist also eine Fläche, die von Bezier-Kurven (Splines) umschlossen wird. Dadurch können viel saubere Übergänge gestaltet und die Treppenbildung wie in der SE1 vermieden werden (allerdings zum Preis einer wesentliche grösseren Komplexität). Ebenfalls bietet die SE2 nebst der horizontalen Punktverschiebung (wie in der SE1)

[27] Weiteres wird im Dokument https://www.vsteno.ch/docs/stenoengines.pdf erläutert.

zwei weitere Möglichkeiten, Zeichen zu neigen: so genannte (1) orthogonale und proportionale Knots (Punkte). Beide Punkte stehen vertikal auf einer Rotationsachse (und rotieren mit ihr), wobei bei den orthogonalen Knots die Abstände vom Rotationspunkt und zur Rotationsachse nicht korrigiert werden (das Zeichen verliert dann durch die Neigung an Höhe, ausserdem können sich gewisse Punkte durch die Drehung unter die Grundlinie verschieben). Proportionale Knots korrigieren den Abstand vom Rotationspunkt (das Zeichen wird bei der Drehung also länger, was z.B. bedeutet, dass der Kopfpunkt eines proportional modellierten Punktes exakt auf der Schreiblinie bleibt). Da bestimmte Teile gewisser Zeichen mehrere Rotationsachsen haben können, offeriert die SE2 auch die Möglichkeit für jeden orthogonalen oder proportionalen Knot eine parallele Rotationsachse zu definieren. Die Funktionalitäten der SE2 funktionieren in VPAINT zu ca. 95%, dennoch ist die SE2 bis heute nicht als PHP-Code innerhalb von VSTENO implementiert.

Linguistischer Teil

In diesem Teil geht es darum, wie Sie mithilfe von VSTENO ein eigenes Stenografie-System definieren können. VSTENO verwendet hierfür eine eigene Formelsprache: Einerseits müssen die Zeichen definiert werden, andererseits benötigt VSTENO auch Regeln, anhand derer es einen Langschrifttest zu einer Folge von Stenogrammen umschreiben kann. Wichtig: Sie müssen nicht programmieren können, um solche Regeln zu erstellen. Allerdings müssen sie die Formelsprache von VSTENO beherrschen: Diese besteht aus einer speziellen Syntax, um die Zeichen zu definieren, sowie einem an REGEX angelehnten Wenn-Dann-Parser (der jeden Text, Wort für Wort und Zeichen für Zeichen umschreibt).

Dies alles wird im Folgenden detailliert und anhand von Beispielen aus dem System Stolze-Schrey (welches der Grundversion von VSTENO standardmässig beiliegt) erklärt. Sie können die Beispiele auch selber ausprobieren, indem Sie auf der Webseite www.vsteno.ch ein Benutzerkonto anlegen und das Modell custom abändern oder neu erstellen.

Benutzerkonto

Gehen Sie auf die Seite www.vsteno.ch und wählen Sie aus der linken Navigationsleiste Konto anlegen. Vervollständigen Sie die Angaben (Benutzername und Passwort müssen mindestens 8 Zeichen lang sein) und lösen Sie das Captcha[28]. Wenn Sie zum ersten Mal ein Konto eröffnen, werden Sie anschliessend direkt eingeloggt und können zwischen dem Stenografie-System (in VSTENO Modell genannt) Standard und Custom wählen. Wählen Sie custom, indem Sie auf den Knopf standard ganz

[28] Falls Sie das System Stolze-Schrey nicht kennen, melden Sie sich bei mir via E-Mail (m.maci@gmx.ch). Gerne erstelle ich Ihnen ein persönliches Konto, mit dem Sie anschliessend ihr eigenes Stenografie-System definieren können. Die Abfrage eines Captchas ist notwendig, damit nur stenokundige Benutzer/innen Zugang zu den Datenbanken erhalten.

links unten klicken. Sie arbeiten nun mit dem custom Modell und können dieses editieren (im Unterschied zum Standard-Modell, das nicht editierbar ist).

Detailliertere Informationen zum Anlegen eines Kontos inklusive grafischer Screenshots finden Sie auch unter: https://www.vsteno.ch/docs/mitmachen_bei_vsteno.pdf.

Stenografische Zeichen

Sie sind also ein/e passionierte/r Stenograf/in, schreiben ein System, das VSTENO noch nicht beherrscht (z.B. DEK, Stiefografie oder Varianten von Stolze-Schrey wie Eil-/Redeschrift oder andere Sprachen wie Französisch, Spanisch, Englisch), und möchten gerne wissen, wie Sie VSTENO dazu bringen, Texte in diesem System auszugeben.

Als erstes müssen Sie VSTENO hierfür beibringen, wie die Zeichen in Ihrem System aussehen. Dies ist durchaus grafisch zu verstehen, das heisst: Sie müssen VSTENO beibringen, welche Linien es schreiben muss, damit ein Zeichen so aussieht, wie es aussehen soll. Auch können Zeichen unter Umständen (je nach System) schattiert werden können.

Dies alles müssen Sie VSTENO beibringen. In gewissem Sinne werden Sie also zu einem/r Stenografie-Lehrer/in - und VSTENO wiederum ist Ihr/e Stenografie-Schüler/in. Das einzige, was Sie nun noch brauchen für Ihren Unterricht, ist die (gemeinsame) Sprache: Also, wie bringen Sie VSTENO bei, wo es wann den Stift ansetzen und wie es ihn bewegen (und wie fest es "drücken") muss, damit die gewünschten Zeichen entstehen?

Vermutlich kennen Sie jene Kindermalbücher mit Zeichnungen, die aus vielen nummerierten Punkten bestehen. Die Punkte dienen dazu, dass ein Kind, das u.U. noch nicht so gut zeichnen kann, relativ simpel einen Elefanten, eine Giraffe (oder was immer vorgegeben ist) zeichnen kann. Die Zeichnung entsteht, indem man den Stift bei Punkt 1 ansetzt und dann alle folgenden Punkte verbindet.

Die gute Nachricht ist nun: VSTENO funktioniert im Prinzip genau gleich! Jedes Zeichen ist als eine "Folge von Punkten" definiert. Der einzige Unterschied zu den Kinderzeichnungen besteht darin, dass VSTENO die Punkte nicht einfach mit einer geraden Linie verbindet, sondern so genannte Bezier-Kurven durch diese Punkte legt - und zwar so, dass die Übergänge je nachdem möglichst "sanft" oder "spitz" verlaufen.

Ein erstes Zeichen ...

Aber der Reihe nach. Beginnen wir mit einem ganz einfachen Zeichen: dem "T" in der Grundschrift Stolze-Schrey. Dieses besteht aus nur einem Strich "von oben nach unten". D.h. wir können zwei Punkte definieren - einen "oberen" und einen "unteren" - und diese dann durch eine gerade Linie verbinden. Der entsprechende Code hierfür sieht folgendermassen aus:

```
"T" => { /*header*/ 6, 0.5, 0, 0, 3, 3, 0, ""
/**/ ,"","","","",0,0,0,0,
/**/ 0,0,0,0,0,0,0,0,
```

```
/*data*/ 0, 20, 0, 1, 1.0, 0, 0, 0,
/**/ 0, 0, 0, 0, 1.0, 0, 1, 0 }
```

Nun gut: Auf den ersten Blick sieht das vielleicht nun doch relativ verwirrlich aus, deshalb vereinfachen wir die Sache gleich noch etwas, indem wir das, was wir für das Verständnis nicht benötigen, weglassen. Ausserdem können Sie alles zwischen den Zeichen /* und */ ignorieren: Hierbei handelt es sich um Kommentare, die für VSTENO keine weitere Bedeutung haben[29]. In diesem Fall weisen die Kommentare /*header*/ und /*data*/ darauf hin, dass es sich bei den folgenden Zahlen um Header-Informationen (= allgemeine Angaben zum Zeichen) und um eigentliche Daten (hier: Punkte oder "Knoten", die das Zeichen definieren) handelt. Wenn wir den Header vorerst mal weglassen, ergibt sich:

```
"T" => { /*data*/ 0, 20, 0, 1, 1.0, 0, 0, 0,
/**/ 0, 0, 0, 0, 1.0, 0, 1, 0 }
```

Damit verbleiben also zwei so genannte Datentupel - eines nach /*data*/ und eines nach /**/, die je aus 8 Werten bestehen. Das erste Datentupel, welches dem ersten Punkt entspricht, enthält die folgenden Werte:

```
 0, 20,  0,  1, 1.0,  0,  0,  0
x1, y1, t1, d1,  th, dr, d2, t2
```

Auf der zweiten Zeile habe ich die Bedeutung der einzelnen Werte notiert. Wichtig sind für uns im Moment vor allem die Werte x1 und y1, welche die Koordinaten des ersten Punktes (0,20) markieren. Der Vollständigkeit halber dokumentieren wir aber gleich alle Bedeutungen[30]:

- x1, y1: x- und y-Koordinate des Punktes
- t1, t2: Spannungen (tensions) im Anschluss an den Punkt (t1) und vor dem folgenden Punkt (t2)[31]
- d1, d2: Art des Punktes (der Wert d1 = 1 bedeutet, dass es sich um einen so genannten entry-point handelt - also den ersten Punkt des Zeichens).
- th: Dicke (thickness) - dieser Wert wird vor allem für Schattierungen verwendet und hat im Moment keine weitere Bedeutung.
- dr: Der so genannte draw-Wert (Zeichnungswert), der bestimmt, ob der Punkt verbunden oder abgesetzt gezeichnet werden soll (der Wert 0 bedeutet, dass der Punkt verbunden wird).

[29] Eine weitere Möglichkeit sind Kommentare mit //: diese sind im Unterschied zu /* und */ jedoch auf eine Zeile beschränkt.

[30] Nur, bitte, tun Sie mir den Gefallen und vergessen Sie Werte, die wir nicht benötigen, gleich wieder; wir haben später Gelegenheit, darauf zurückzukommen!

[31] Die Bedeutung der tension wird später erklärt. Sie gibt bei einer Bezier-Kurve an, ob der Punkt "spitz" oder "sanft" (rund) verbunden werden soll. Der Wert 0 gibt hier an, dass die Verbindung spitz sein soll, wie es das Zeichen "T" verlangt.

Es mag sein, dass die vielen Informationen Ihnen im Moment wie etwas viele Bäume im Wald vorkommen, aber wie bereits erwähnt geht es im Moment nur um die x- und y-Koordinaten. Wie man sehen kann wird im ersten Datentupel der Punkt 1=(0,20) und im zweiten Datentupel der Punkt 2=(0,0) definiert. Dies bedeutet nichts anderes als die Umsetzung dessen, was wir weiter oben vermerkt haben: Das Zeichen "T" ist die Verbindung zwischen einem "oberen" Punkt (0,20) und einem "unteren" Punkt (0,0).

An dieser Stelle weise ich gleich auf zwei weitere wichtige Aspekte hin: (1) Stenografie-Zeichen werden senkrecht (also ohne Neigung) definiert[32] und (2) das Koordinatensystem von VSTENO verläuft auf der x-Achse von links nach rechts und auf der y-Achse von unten nach oben. Die Grösse der Zeichen können Sie als Linguist/in im Prinzip frei wählen (da es sich um Vektorkoordinaten handelt, lassen sich die Zeichen später beliebig und verlustfrei vergrössern oder verkleinern). Es wird aber empfohlen, eine gut lesbare und intuitiv verständliche Standardgrösse zu verwenden. Im vorliegenden Fall verwenden wir 10 Punkte für eine Stufe des Systems Stolze-Schrey. Da das Zeichen "T" zwei Stufen hoch ist, ergibt dies für die y-Koordinate den Wert 20. Beachten Sie, dass alle Werte Fliesskommazahlen sind, d.h. Sie können auch Koordinaten mit Kommastellen - z.B. 19.5 oder 19.999999 - verwenden.

Wenn Sie sehen möchten, wie das Zeichen "T" aussieht, dann gehen Sie am besten zur Webseite www.vsteno.ch und geben Wörter mit "T" ein, z.B. "beten". Standardmässig ist eine Neigung von 60° voreingestellt, weshalb das Zeichen nicht senkrecht, sondern um 60 Grad geneigt erscheint. Wenn Sie das Zeichen so sehen möchten, wie wir es oben definiert haben, dann geben Sie im Formular unter Optionen 90° ein: Sowohl der Buchstabe "B" als auch "T" werden nun senkrecht dargestellt. Machen Sie sich im Moment noch keine Gedanken über den Vokal "E" oder die Endkürzung "EN". Widmen wir uns nun als nächstes dem Buchstaben "B". Dieser weist im Unterschied zu "T" eine Rundung am unteren Ende auf.

Rund oder spitzig?

Im ersten Abschnitt haben wir den denkbar einfachsten Fall behandelt: ein Zeichen, das mit zwei Punkten definiert wird, die spitzig miteinander verbunden werden sollen. Die Art und Weise, wie die Punkte verbunden werden sollen, ist im Datenfeld-Tension definiert. Betrachten wir noch einmal die zwei Punkte von "T":

```
"T" => { 0, 20, /*p1t1*/ 0, 1, 1.0, 0, 0, /*p1t2*/ 0,
/**/ /*p2t1*/ 0, 0, 0, 0, 1.0, 0, 1, /*p2t2*/ 0 }
```

Zur Veranschaulichung habe ich Kommentare eingefügt. Wie man sieht, enthält jeder Punkt (p1 und p2) zwei Tensions (t1 und t2). Bevor wir über die Bedeutung dieser Tensions weiterreden, empfehle ich Ihnen am besten, die folgende Internet-Seite zu besuchen: Sie enthält eine interaktive Demo so genannter Splines. Ein Spline ist nichts anderes als eine Folge von Punkten (wie wir sie in unseren Zeichen,

[32] Wenn sie, wie im System Stolze-Schrey, geneigt sein sollen, so kann VSTENO das geneigte Zeichen anhand des senkrechten selbständig berechnen.

z.B. "T", definieren). Sie können nun mit den Tension der verschiedenen Punkte (im Englischen manchmal auch knots genannt) herumspielen und so ein intuitives Verständnis dafür bekommen, welchen Einfluss die Spannung auf den Verlauf von Bezier-Kurven haben:

http://scaledinnovation.com/analytics/splines/aboutSplines.html

Vereinfacht gesagt ergibt die Spannung mit dem Wert 0 eine spitze Verbindung, die Spannung mit dem Wert 0.5 eine runde Verbindung (es sind natürlich auch andere Werte - also "stärkere" und "schwächere" Spannungen - möglich). Wichtig zu wissen ist, dass die Kurve zwischen zwei Punkten P1 zu P2 durch zwei Spannungen definiert wird: Die Spannung p1t1 gibt die Spannung nach dem Punkt P1 (Richtung P2) an, die Spannung p1t2 gibt die Spannung vor dem Punkt P2 (aus Richtung P1) an. Wir verwenden diese Spannungen nun, um das Zeichen "B" zu definieren:

```
"B" => { /*data p1*/ 0, 10, 0, 1, 1.0, 0, 0, 0.5,
/*p2*/ 0, 2, 0.5, 0, 2.5, 0, 0, 0.5,
/*p3*/ 2.5, 0, 0.5, 0, 1.0, 0, 0, 0.5,
/*p4*/ 5, 2, 0.5, 0, 1.0, 0, 1, 0 }
```

Wiederum haben wir aus Gründen der Übersichtlichkeit den Header weggelassen. Wenn wir nur die Koordinaten aus den Tupeln anschauen, wird das Zeichen "B" mit 4 Punkten definiert: P1(0,10), P2(0,2), P3(2.5,0), P4(5,2). Zur Veranschaulichung können Sie z.B. ein Blatt Papier nehmen, die Punkte auf einem Koordinatensystem eintragen und diese dann - wie eingangs anhand der Kinderzeichnungen erwähnt - verbinden. Beachten Sie, dass dieses Zeichen nur 10 Punkte hoch ist (im Unterschied zu "T" ist "B" einstufig).

Der ganze mathematische Hokuspokus - bzw. die Magie ;-) - liegt nun in den Tensions. Sie betragen für die Punkte 1-4: T1(0,0.5), T2(0.5,0.5), T3(0.5,0.5), T4(0.5,0). Das bedeutet, dass beim Zeichen "B" nur der erste Punkt spitz (Wert 0) verbunden wird. Alle anderen Punkte werden danach rund (mit der Spannung 0.5) verbunden. Die letzte Spannung T4 enthält in diesem Beispiel wieder den Wert 0: Dieser hat keine Bedeutung, denn wenn wir das Zeichen "B" mit einem weiteren Zeichen verbinden, so können wir nicht wissen, ob die Verbindung mit dem folgenden Zeichen rund oder spitzig sein muss (im Falle von "T" spitzig, im Falle von "M" hingegen rund). Anders gesagt: Dieser Wert hängt vom Folgezeichen ab - und es ist somit völlig egal, welchen Wert Sie hier eintragen (er wird später überschrieben).

Schattierungen

Bis jetzt haben wir Zeichen als Folge von Punkten definiert, die spitz oder rund miteinander verbunden werden. Damit lässt sich schon einiges machen! Allerdings verlangen gewisse Stenografie-Systeme - darunter auch Stolze-Schrey -, dass man Zeichen schattieren kann. Auch diese Funktion können wir durch Setzen der entsprechenden Werte innerhalb der Datentupel erreichen. Nehmen wir noch einmal unser einfaches Zeichen "T", wie wir es definiert haben:

```
"T" => { /*data p1*/ 0, 20, 0, 1, 1.0, 0, 0, 0,
/*p2*/ 0, 0, 0, 0, 1.0, 0, 1, 0 }
```

An der 5. Stelle sehen wir hier den Wert 1.0. Dieser Wert entspricht der Dicke (thickness) und bedeutet also, dass das Zeichen immer mit der Standarddicke gezeichnet werden soll. Genau genommen ist der Wert 1.0 ein Multiplikationsfaktor in Bezug zu einer (vom/von der Nutzer/in vorgegebenen) Grunddicke: Werte >1.0 geben eine dickere, Werte <1.0 eine dünnere Linie an. Es können - wie für alle anderen Werte - Fliesskommazahlen verwendet werden (also auch 0.77 ist eine gültige Liniendicke). Wir passen diesen Wert nun an:

```
"T" => { /*data p1*/ 0, 20, 0, 1, 2.5, 0, 0, 0,
/*p2*/ 0, 0, 0, 0, 1.0, 0, 1, 0 }
```

Wie man sieht, habe ich im ersten Datentupel (= Punkt 1) als Dicke den Wert 2.5 eingetragen. Dies bedeutet nun Folgendes: Wenn das Zeichen schattiert werden soll, dann wird ausgehend vom ersten Punkt P1 die Linie (genauer: Bezier-Kurve) zum Punkt P2 mit der 2.5-fachen Dicke gezeichnet. Bitte beachten Sie: Für das Zeichen "T" genügt es, nur diesen einen Wert zu ändern, um das Zeichen zu schattieren. Würden wir auch im Punkt P2 den Wert auf 2.5 erhöhen, so würde dies bedeuten, dass auch die Verbindungslinie zum nächsten Zeichen schattiert würde (was wir nicht wollen). Sie können diese Schattierung sehen, indem Sie in der Demoversion das Wort "Tat" eingeben: Das erste "T" wird normal gezeichnet, das zweite schattiert.

Leider ist es aber nicht immer so einfach mit den Schattierungen wie beim Zeichen "T", das mit einem spitzen Punkt beginnt und endet. Speziell bei Zeichen, die mit Rundungen beginnen und/oder aufhören, sieht es unschön aus, wenn wir einfach ab einem bestimmten Punkt eine wesentlich dickere Linie definieren. Bei runden Zeichen empfiehlt es sich also, die Schattierung in mehrere Schritte abgestuft beginnen und/oder enden zu lassen. Zur Illustration zeigen wir dies an unserem zweiten Beispielzeichen "B", welches einen spitzen Anfang und ein rundes Ende aufweist:

```
"B" => { /*data p1*/ 0, 10, 0, 1, 2.5, 0, 0, 0.5,
/*p2*/ 0, 2, 0.5, 0, 1.75, 0, 0, 0.5,
/*p3*/ 2.5, 0, 0.5, 0, 1.0, 0, 0, 0.5,
/*p4*/ 5, 2, 0.5, 0, 1.0, 0, 1, 0 }
```

Da unser Zeichen mit einem spitzen Punkt beginnt, können wir hier problemlos direkt die Dicke 2.5 einsetzen. Die Kurve von P1 zu P2 wird also "maximal dick" gezeichnet. Danach setzen wir ab Punkt P2 eine mittlere Dicke von 1.75 ein. Die Kurve von P2 zu P3 wird also "weniger dick" gezeichnet. Schliesslich bleibt noch die Verbindung von P3 zu P4: Hier kehren wir zur "normalen" Dicke von 1.0 zurück.

Intermediate shadow points

Die soeben dargestellte Abstufung der Schattierungen funktioniert relativ gut, wenn das Zeichen selbst bereits aus mehreren Punkten besteht, die eine abgestufte Definition der Schattierung zulassen. Schwieriger wird es, wenn das Zeichen als solches

zu wenige Punkte enthält, um eine optisch einigermassen gelungene Abstufung zu erreichen. In diesem Fall besteht zwar die Möglichkeit, zusätzliche Punkte in die Zeichendefinition einzufügen, um mehr Zwischenschritte in der Schattierung zu erhalten. Diese Zwischenpunkte können jedoch den Nachteil haben, dass sie das optische Bild in der unschattierten Variante stören, da der kontinuierliche Lauf der Bezier-Kurve unterbrochen wird. Die von VSTENO verwendete Lösung besteht hier in so genannten "intermediate shadow points". Dies sind Zwischenpunkte, die nur dann gezeichnet werden, wenn das Zeichen schattiert dargestellt werden soll. In der normalen Zeichendarstellung werden diese Punkte ignoriert. Als Beispiel zeigen wir das Zeichen "R"[33]:

```
"VR" => { /*data*/  /*1*/ 2.5, 5, 0.5, 1, 1.5, 0, 0, 0.5,
/*2*/ 3.75, 4, 0.7, 5, 2.5, 0, 0, 0.7,
/*3*/ 5, 2.5, 0.7, 0, 3.0, 0, 0, 0.7,
/*4*/ 4.5, 0.5, 0.7, 5, 2, 0, 0, 0.7,
/*5*/ 3.25, 0.15, 0.7, 5, 1.5, 0, 0, 0.7,
/*6*/ 2.5, 0, 0.7, 0, 1.0, 0, 0, 0.5,
/*7*/  0, 2.5, 0.7, 0, 1.0, 0, 0, 0.5,
/*8*/ 2.5, 5, 0.5, 0, 1.0, 0, 1, 0.0 }
```

Der Einfachheit halber wurden die Punkte 1-9 innerhalb von Kommentaren nummeriert. Für die Definition des nicht schattierten Zeichens "R" reichen im Prinzip die folgenden Punkte aus: 1=(2.5,5), 3=(5,2.5), 6=(2.5,0), 7=(0,2.5), 8=(2.5,5). Wie man sehen kann, markieren diese 5 Punkte, die Eckpunkte eines geschlossenen Kreises, die jeweils rund - d.h. mit Tensions zwischen 0.5-0.7 - verbunden werden. Würde man allerdings nur diese 5 Punkte für das schattierte Zeichen verwenden, so stünden für die Schattierung lediglich die Strecken von Punkt 1 - 3 und von Punkt 3 - 6 zur Verfügung. Aus diesem Grund wurden die übrigen Punkte - also 2, 4, 5 - als intermediate shadow points eingefügt. Erkennbar ist dies am Wert 5 an der 4. Stelle im Datatupel. Wie wir weiter oben schon angedeutet hatten, steht der Wert in der 4. Position für den "Typ" des Punktes. Wir haben hier schon den entry point (mit Wert 1) und den normalen Punkt (mit Wert 0) kennen gelernt. Der intermediate shadow point ist also einfach ein weiterer Typ, den ein Punkt annehmen kann.

Wie bereits erwähnt würde das Zeichen "VR" unschön aussehen, wenn diese Zwischenpunkte auch für das unschattierte Zeichen verwendet werden (dies ist hier besonders deutlich, weil Kreise kontinuierliche Kurven sind, wo es besonders auffällt, wenn sie durch weitere Punkte unterbrochen werden). Aus diesem Grund lässt der Zeichenalgorithmus von VSTENO diese Zusatzpunkte weg, wenn das Zeichen nicht schattiert ist.

[33] Da der Laut "R" im System Stolze-Schrey zwei Ausführungsvarianten hat, unterscheiden wir zwischen "AR" (= Anlaut-R) und "VR" (= vokalisches R). Das "vokalische R" ist somit jenes, das im Uhrzeigersinn ausgeführt wird und nach Vokal steht.

Punkttypen

An dieser Stelle ist es nun an der Zeit, die verschiedenen Punkttypen in ihrer Vollständigkeit vorzustellen. Wiederum: Vergessen Sie bitte sämtliche Punkttypen, die hier vorgestellt werden und die noch nicht anhand von Beispielen erklärt werden. Im Moment genügt es, wenn Sie ein erstes Mal gehört haben, dass diese Punkttypen existieren.

Wichtig im Zusammenhang mit diesen Punkttypen ist zu wissen, dass es im Datentuplet zwei Stellen gibt, wo diese definiert werden. Hier noch einmal das Datentuplet des ersten Punktes, das wir für das Zeichen "T" definiert hatten:

```
0,  20,  0,  1, 1.0,  0,  0,  0
x1, y1, t1, d1,  th, dr, d2, t2
```

Das Feld d1 steht für die Eingangs-Information (entry information), d.h. die Information, die angibt, wie das Zeichen verbunden wird. Es kann folgende Werte annehmen:

Wert	Bedeutung
0	normaler Punkt
1	entry point (Anfangspunkt = erster Punkt des Zeichens)
2	pivot point (Drehpunkt)
3	conditional pivot point (bedingter Drehpunkt)[34]
4	connecting point (Verbindungspunkt)
5	intermediate shadow point (Zwischenschattierungspunkt)
98	late entry point (später Anfangspunkt)

Analog steht das Feld d2 für die Ausgangs-Information (exit information), d.h. die Information, die angibt, wie das Zeichen "beendet" wird. Die Werte sind ähnlich wie bei d1, aber mit einigen kleinen Unterschieden:

Wert	Bedeutung
0	normaler Punkt
1	exit point (Endpunkt = letzter Punkt des Zeichens)
2	pivot point (Drehpunkt)
3	conditional pivot point (bedingter Drehpunkt)[35]
99	early exit point (früher Endpunkt)

Wie gesagt: Zerbrechen Sie sich nicht jetzt schon den Kopf über alle diese Punkttypen, wir werden sie einzeln mit Beispielen erläutern. Beginnen werden wir mit dem pivot point, also dem "Drehpunkt", mit dem Wert 2 in d1 oder d2.

Verbundene Rundungen

Zwei Stenozeichen mit spitzen Anfangs- und Endpunkten zu verbinden, ist trivial: Eine gerade Linie genügt! Schwieriger wird es jedoch, wenn eines - oder beide Zeichen - Rundungen als Anfangs- oder Endpunkt aufweisen. Möglich sind hier die Kombinationen spitz + rund ("T" + "M" im Wort "Thema"), rund + spitz ("B" + "T" im Wort "beten") oder rund + rund ("M" + "M" im Wort "Mumie"). Zusätzlich kann im System Stolze-Schrey (das wir immer als Beispiel nehmen) das folgende

Zeichen eng oder weit verbunden und hoch oder tiefgestellt werden. Während die Verbindung "eng" vs "weit" noch einigermassen überschaubar ist (2 Möglichkeiten) ergibt sich bei der Hoch- und Tiefstellung eine Vielzahl von Fällen: Zum einen bedeutet Hoch- und Tiefstellung nicht bei jedem Zeichen dasselbe (beim Zeichen "B" bedeutet Hochstellung eine halbe Stufe höher, beim Zeichen "SCH" hingegen beträgt der Unterschied eine ganze Stufe), zum anderen können Zeichen halb-, ein-, zwei- oder dreistufig sein, was zu einer Unzahl von Verbindungsarten (damit meinen wir vor allem Länge, Winkel und allenfalls Verlauf der Verbindung) führt.

Aus diesem Grund besteht die Möglichkeit bei gerundeten Zeichen einen Drehpunkt zu definieren. Dieser Drehpunkt sollte sich im Fuss- oder Scheitelpunkt des Zeichens befinden und somit so gelegen sein, dass er das Zeichen in 2 (Zeichen mit einer Rundung) oder 3 Teile (Zeichen mit 2 Rundungen) auftrennt. D.h. dass solche Zeichen über einen fixen Mittel- oder Kernteil verfügen, an den sich die variablen Rundungen anschliessen. Der Drehpunkt markiert also den Übergang zwischen dem fixen und dem Variablen Teil eines Zeichens. Wir illustrieren dies am Beispiel "B", welches eine Rundung am Fusse aufweist:

```
"B" => { /*data*/ /*1*/0, 10, 0, 1, 1.0, 0, 0, 0.5,
/*2*/ 0, 2, 0.5, 0, 2.5, 0, 0, 0.5,
/*3*/ 2.5, 0, 0.5, 0, 1.0, 0, 2, 0.5,
/*4*/ 5, 2, 0.5, 0, 1.0, 0, 1, 0 }
```

Diese Definition ist identisch mit jener, die wir bereits weiter oben gegeben haben, mit dem einzigen Unterschied, dass der 3. Punkt als Drehpunkt definiert wird (Wert 2 im vorletzten Datentupel). Dies bedeutet nun, dass die Punkte 1-3 fix (also unveränderlich) sind, während der Punkt 4 (= exit point) variabel ist. Betrachten wir hier das Beispiel "Bohne": Hier trifft die Rundung von "B" mit der Tiefstellung des einstufigen Zeichens "N" zusammen, welches eng verbunden wird. Wie man leicht erkennen kann, befindet sich somit der Anschlusspunkt des Zeichens "N" unterhalb (!) des Endpunktes des Zeichens "B" (in Koordinaten gesprochen: der Endpunkt von "B" liegt bei y=2, der Anschlusspunkt von "N" liegt bei y=0). Mit anderen Worten: Würde das Zeichen "B" ohne Anpassung verbunden, dann entstünde ein hässlicher "Schnörkel" anstelle einer kontinuierlichen Linie.

VSTENO passt in diesem Fall die Endpunkte beider Zeichen (auch "N" ist ein Zeichen, welches mit einer Rundung beginnt) an, sodass ein "sanfter" Übergang entsteht.[36]

Early exit und late entry

Und es geht weiter mit einem neuen Punkttyp: dem early exit point oder dem vorzeitigen Endpunkt. Dieser wird im System Stolze-Schrey für Zeichen mit Unterschlaufe verwendet, die am Ende eines Wortes ohne Schlaufe geschrieben werden. Beispiele sind hier die Zeichen "NG" oder "NS" (als einstufige Variante) oder "SCH", "CH",

[36] Die Anpassung erfolgt im Moment als simple Gerade, d.h. VSTENO nimmt den Drehpunkt von "B" und den Drehpunkt von "N", zieht eine Gerade zwischen den beiden und passt den Endpunkt von "B" und den Anfangspunkt von "N" so an, dass sie auf dieser Geraden liegen.

"SCHW", "ZW" usw. (als zweistufige Variante). Der early exit point bedeutet also nichts anderes, als dass das Zeichen an diesem Punkt "vorzeitig" endet, wenn es am Ende eines Wortes steht. Hier das Zeichen "NS":

```
"NS" => { /*data*/ 0.75, 5, 0.5, 1, 1.5, 0, 0, 0.5,
/**/ 3.75, 8.5, 0.5, 2, 2.0, 0, 0, 0.5,
/**/ 2.65, 10, 0.5, 0, 2.5, 0, 0, 0.5,
/**/ 1.75, 9, 0.5, 0, 3.0, 0, 0, 0.5,
/**/ 1.75, 1, 0.5, 0, 3.0, 0, 0, 0.5,
/*late entry point*/ 0.75, 0, 0.5, 0, 2.5, 0, 99, 0.5,
/**/ 0, 2.25, 0.5, 0, 1.5, 0, 2, 0.5,
/**/ 1.75, 3, 0.5, 0, 1.0, 0, 1, 0.5 }
```

Einen ähnlichen Fall gibt es bei Zeichen, die mit einer Rundung beginnen und wo in Verbindung mit anderen Zeichen ein Teil der Rundung zweimal gezeichnet wird. Nehmen wir als Beispiel das Zeichen "V": In Zusammensetzungen wie "davon" - wo das Zeichen vom Fusspunkt der Kürzung "DA" - von unten her verbunden werden muss, wird der oberste Teil des Zeichens "V" zweimal gezeichnet (einmal beim Hochfahren, einmal beim Herunterfahren). Dies ist in diesem Fall - d.h. wenn das Zeichen verbunden ist - richtig und somit kein Problem. Nehmen wir nun aber nur die Kürzung "VON" (welche dem Zeichen "V" alleine entspricht). Auch hier würde VSTENO den obersten Teil des Zeichens zweimal zeichnen - was nicht schön ist (wenn die beiden Kurven beim Hoch- und Herunterfahren nicht exakt übereinander liegen, dann erscheint das Kopfende dunkler). Deshalb existiert auch hier der late entry point oder eben der "späte Eintrittspunkt":

```
"V" => { /*data*/ 1, 16, 0.5, 1, 1.0, 0, 0, 0.5,
/**/ 2, 18, 0.6, 2, 1.0, 0, 0, 0,
/*late entry point*/ 6, 20, 0, 98, 0, 0, 0, 0.5,
/**/ 2, 18, 0.6, 0, 1.5, 0, 0, 0.5,
/**/ 1, 16, 0.5, 0, 2.5, 0, 0, 0.5,
/**/ 0, 14, 0.5, 0, 3.0, 0, 0, 0.5,
/**/ 0, 5, 0.5, 0, 3.0, 0, 0, 0.5,
/**/ 1, 2, 0.5, 0, 2.5, 0, 0, 0.5,
/**/ 3, 0, 0.5, 4, 1.5, 0, 0, 0.5,
/**/ 3, 0, 0.5, 0, 1.5, 0, 2, 0.5,
/**/ 5, 2, 0.5, 0, 1.0, 0, 1, 0.5 }
```

Der Wert 98 bedeutet hier also, dass die Punkte 1 und 2 nicht gezeichnet werden, wenn das Zeichen am Anfang steht. In diesem Fall wird der late entry point (also der 3. Punkt) als Anfangspunkt gesetzt.

Header

Bei all unseren Definitionen haben wir bis jetzt den Header der Einfachheit halber weggelassen. Dieser enthält Informationen, die das ganze Zeichen (also nicht nur

einzelne Punkte) betreffen. In der aktuellen Version ist der Header 24 Felder lang, welche folgende Bedeutungen haben:

Offset	Bedeutung
0	token width (Zeichenbreite)
1	delta-y before (Delta-Y vorher)
2	delta-y after (Delta-Y nachher)
3	tension before (Spannung vorher)
4	additional x before (zusätzliches x vorher)
5	additional x after (zusätzliches x nachher)
6	additional delta-y (zusätzliches Delta-Y)
7	*rev1: 2nd parallel rotating axis[37]
8	*rev1: 3rd parallel rotating axis
9	*rev1: 4th parallel rotating axis
10	nicht verwendet
11	nicht verwendet
12	token type (Zeichentyp)
13	inconditional delta-y before (unbedingtes Delta-Y vorher)
14	inconditional delta-y after (unbedingtes Delta-Y nachher)
15	alternative exit point x (alternativer Endpunkt x)
16	alternative exit point y (alternativer Endpunkt y)
17	exit point to use (zu verwendender Endpunkt)
18	interpretation y coordinates (Interpretation y-Koordinaten)
19	vertical (vertikale Höher-/Tieferstellung)
20	distance (horizontaler Abstand)
21	shadowed (schattiert)
22	don't connect (nicht verbinden)
23	token group

Beachten Sie bitte, dass wir die Feldnummer als Offset bezeichnen und mit der Nummerierung bei 0 beginnen. Auch hier werden wir wieder Schritt für Schritt vorgehen und die Bedeutung der einzelnen Werte anhand von Beispielen zeigen. Wir beginnen mit den Feldern 19-21, welche die Höher-/Tieferstellung (Offset 19), den horizontalen Abstand (Offset 20) und die Schattierung (Offset 21) betreffen.

Vokale

Diese drei Felder können wir verwenden, um im System Stolze-Schrey die Vokale zu definieren. Im Unterschied zu Konsonantenzeichen entsprechen diese nämlich keinen real geschriebenen Zeichen, sondern werden durch die Art, wie Zeichen verbunden werden, dargestellt. Mit anderen Worten: Ein Vokal entspricht im System Stolze-Schrey einem "leeren Zeichen" (= keine Punkte im Data-Bereich), das nur aus einem Header besteht. Als Beispiel der Vokal "E", der aus einer weiten Verbindung ohne Hochstellung und ohne Schattierung besteht:

```
"E" => { /*header0-7*/ 0, 0, 0, 0, 0, 0, 0, "",
/*8-15*/ "","","","", 2, 0, 0, 0,
```

```
/*16-23*/ 0,0,0,"no","wide","no",0,0,
/*data*/ }
```

Zunächst einmal weisen wir auf den Offset 12 hin, der dem Zeichentyp (token type) entspricht. Hier wurde der Wert 2 gesetzt, der "virtuelles Zeichen" bedeutet (dagegen bezeichnet der Wert 0 ein "normales" und der Wert 1 ein "unbedingt schattiertes" Zeichen, wie wir später sehen werden). "Virtuell" bedeutet in diesem Fall nichts anderes, als dass das Zeichen keine grafische Entsprechung (als "Punktezeichnung") hat, sondern nur aus einem Header besteht.

Weiter weisen wir im Header-Untertupel 16-23 auf die Werte "no" (Offset 19), "wide" (Offset 20) und "no" (Offset 21) hin. Diese entsprechen somit der Angabe "keine Höher-/Tieferstellung" (also horizontale Verbindung), "weite Verbindung" und "nicht schattiert". Anbei die Liste der Werte, die Felder annehmen können:

Offset	Wert & Bedeutung
19	"no" (horizontale Verbindung), "up" (Höherstellung), "down" (Tieferstellung)
20	"wide" (weite Verbindung), "narrow" (enge Verbindung), "no" (kein Abstand)
21	"no" (keine Schattierung), "yes" (Schattierung)

Ein weiteres Beispiel: Der Diphtong "AU" wird als "enge Verbindung", "hochgestellt" und "schattiert" definiert:

```
"AU" => { /*header0-7*/ 0, 0, 0, 0, 0, 0, 0, "",
/*8-15*/ "","","","", 2, 0, 0, 0,
/*16-23*/ 0,0,0,"up","narrow","yes",0,0,
/*data*/ }
```

Bitte beachten Sie, dass in beiden Fällen der Data-Bereich leer bleibt: Die Vokale enhalten also wie bereits erwähnt keine Punkte!

Hochstellung

Mit den Offsets 19-21 im Header können wir also die Verbindung von Zeichen und insbesondere deren Hoch- oder Tiefstellung (Offset 19) bestimmen. Allerdings bedeutet "Hochstellung" nicht in allen Fällen dasselbe: Bei den meisten Zeichen im System Stolze-Schrey bedeutet es, dass das Folgezeichen eine halbe Stufe höher (d.h. dass sich die y-Koordinate sich um den Wert 5 erhöht) geschrieben wird. Es gibt jedoch auch Zeichen - z.B. "SCH", "CH", "Z", "ZW" etc. -, welche eine ganze Stufe (als 10 Punkte) höher geschrieben werden müssen. Ausserdem unterscheiden sich diese Zeichen darin, wie ein weiteres Folgezeichen angeschlossen wird: Bei den meisten Zeichen werden weitere Folgezeichen ebenfalls eine halbe Stufe höher geschrieben, bei "SCH", "CH", "Z", "ZW" etc. hingegen, muss nach der Hochstellung wieder zur Grundlinie zurückgekehrt werden. All dies kann im Header mit den Offsets 1 (delta-y before) und 2 (delta-y after) definiert werden:

```
"SCH" => { /*header0-7*/ 9, 1,-1, 0.5, 0, 0, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 0, 0, 0, 0, 0, 0, 0,
/*data*/ /*...*/ }
```

Hier wird also im Offset 1 ein delta-y before mit dem Wert 1 (= 1 Stufe höher) und im Offset 2 ein delta-y after mit dem Wert -1 (= eine Stufe tiefer) definiert. Mit anderen Worten: Nachdem das Zeichen "SCH" eine Stufe höher an das Vorzeichen angeschlossen wird, wird die Schreiblinie danach wieder um den Wert -1 auf die ursprüngliche Linie zurückgesetzt. Im Vergleich dazu das Zeichen "B":

```
"B" => { /*header0-7*/ 5, 0.5, 0, 0, 1, 1, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 0, 0, 0, 0, 0, 0, 0,
/*data*/ /*...*/ }
```

Hier wird die Schreiblinie vor dem Zeichen um eine halbe Stufe (Wert 0.5) erhöht und danach gleich belassen (der Wert 0 bedeutet, dass keine Veränderung vorgenommen wird).

Zeichenbreite

Bis jetzt haben wir Zeichen definiert, ohne uns Gedanken über deren Breite zu machen. Das sollten wir nun tun, denn es ist offensichtlich, dass jedes Zeichen eine Breite hat bzw. haben muss, um eine vernünftige und saubere Aneinanderreihung zu erreichen. Betrachten wir zum Beispiel noch einmal unser allererstes Zeichen "T", das aus einem einzigen senkrechten Strich besteht:

```
"T" => { /*header0-7*/ 6, 0.5, 0, 0, 3, 3, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 0, 0, 0, 0, 0, 0, 0,
/*data p1*/ 0, 20, 0, 1, 1.0, 0, 0, 0,
/*p2*/ 0, 0, 0, 0, 1.0, 0, 1, 0 }
```

Da dieses Zeichen nur aus einem senkrechten Strich besteht, der in einer bestimmten Dicke gezeichnet wird, ist es per se nur so breit wie der Strich dick ist. Das macht aber wenig Sinn, da das Zeichen so unmittelbar auf ein vorhergehendes oder ein nachfolgendes Zeichen angeschlossen wird. Selbst bei "engem" bzw. "keinem" Abstand zwischen den Zeichen, sollten sie ein Minimum voneinander entfernt sein. Dies können wir durch die Werte in den Offsets 4 (additional x before) und Offset 5 (additional x after), sowie dem Offset 1 (token width) erreichen. Der Wert in Offset 4 entspricht dabei dem "linken", Offset 5 dem "rechten" Leerraum, der auf das Zeichen folgt. Im obigen Beispiel wurde hier 3 Pixel für den linken und 3 Pixel für den rechten Abstand definiert, was für das Zeichen eine Gesamtbreite von 6 Pixeln (Offset 0) ergibt.

Bitte beachten Sie, dass die Werte in den Offsets 4 und 5 zusätzlich zur jener Breite addiert werden muss, welche sich aus den Punkten, die das Zeichen definieren, ergibt:

```
"Y" => { /*header0-7*/ 14, 0.5, 0, 0, 2, 2, 0, ""
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 0, 0, 0, 0, 0, 0, 0,
```

```
/*data p1*/ 0, 10, 0, 1, 3.0, 0, 0, 0,
/*p2*/ 10, 0, 0, 0, 1.0, 0, 1, 0 /**/ }
```

Das Zeichen "Y" - in Stolze-Schrey als gerader Strich von links oben P1(0,10) nach rechts unten P2(10,0) definiert, ist - rein von den Punkten her - bereits 10 Pixel breit. Wird nun in Offsets 4 und 5 ein zusätzlicher Leerabstand links und rechts von 2 Pixeln (also insgesamt 4 Pixel) definiert, so ist das Zeichen insgesamt 14 Pixel breit.

Es empfiehlt sich, beim Erstellen neuer Zeichen, mit diesen Möglichkeiten etwas herumzuspielen, um optimale Werte zu finden, die in Verbindungen mit anderen Zeichen am besten aussehen[38].

Schreiblinienverschiebung

Es gibt auch Zeichen, bei welchen sich die Schreiblinie in jedem Fall - also unabhängig von Hoch- oder Tiefstellung - verschiebt. Im System Stolze-Schrey ist dies z.B. bei "RR" der Fall: Hier müssen Folgezeichen eine Stufe höher angeschlossen werden. Wir erreichen dies durch Setzen der Offsets 13 (inconditional y before) und 14 (inconditional y after):

```
"RR" => { /*header0-7*/ 10, 0.5, 0, 0.5, 0, 0, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 0, 0, 0, 0, 0, 0, 0,
/*data p1*/ 1, 7.75, 0.5, 1, 1.0, 0, 0, 0.5,
/*p2*/ 5, 10, 0.7, 0, 3.0, 0, 0, 0.8,
/*p3*/ 10, 5, 0.8, 0, 3.0, 0, 0, 0.7,
/*p4*/ 5, 0, 0.7, 0, 1.0, 0, 0, 0.5,
/*p5*/ 0, 5, 0.5, 0, 1.0, 0, 0, 0.5,
/*p6*/ 1, 7.75, 0.5, 0, 1.0, 0, 0, 0.5,
/*p7*/ 5, 10, 0.5, 0, 1.0, 0, 1, 0.5, /**/ }
```

In diesem Zeichen sehen wir gleich zwei Phänomene: (1) Die Schreiblinie verschiebt sich nach dem Zeichen in Normalstellung um 1 Stufe nach oben (Offset 14 mit Wert 1) und (2) Wird das Zeichen höher gestellt, so beträgt die Höherstellung eine halbe Stufe (Wert 0.5 im Offset 1) und die Schreiblinie liegt 1.5 (d.h. die Summe aus Offset 1 und Offset 14) höher![39]

[38] Im Laufe der Entwicklung der SE1 hat sich gezeigt, dass dieses Konzept der fixen Abstände (mit Vor/Nachbreite der Zeichen) ungenügend ist. Es ist zwar sinnvoll für Zeichen, die effektiven einen fixen Abstand haben (wie z.B. Buchstaben in handschriftlicher Blockschrift, die mit VSTENO ebenfalls umgesetzt werden können). Stenografie-Zeichen benötigen jedoch je nach Zeichen und Verbindung (eng/weit, hoch/tief) verschiedene Abstände. Dies wurde gelöst, indem für Stenografie-Zeichen prinzipiell nur noch eine "Nettobreite" (Offset 0) und keine Vor/Nachbreite (Offsets 4/5) mehr angegeben wird. Die individuellen Abstände werden danach via REGEX-Regeln eingesetzt. Damit dies funktionierte, musste ein weiterer Zeichentyp (Offset 12 im Header) definiert werden. Der Wert 3 in Offset 12 bedeutet somit "spacer": VSTENO wird in diesem Fall nur die Breite (Offset 0) dieses Zeichens berücksichtigen. Zu sehen ist das in den Spacer-Zeichen #1, #2, #3, ..., #9 in https://github.com/marcelmaci/vsteno/blob/master/ling/grundschrift_stolze_schrey_redesign.txt.

[39] Falsch wäre hier, die Schreiblinienverschiebung in Offset 3 - z.B. mit dem Wert 0.5 - zu definieren: Dies würde zwar für höher gestellte "RR" gelten, normal gestellte "RR" würden jedoch

Unbedingte Schattierung

Und weiter geht's mit Besonderheiten des Systems Stolze-Schrey. Nebst allen präsentierten Anforderungen an stenografische Zeichen, gibt es auch Abkürzungen, in jedem Fall eine Schattierung verlangen. Ein Beispiel dafür ist die Abkürzung "AUCH", welche aus einem höher gestellt, schattierten "CH" besteht:

```
"AUCH" => { /*header0-7*/ 5, 1,-1, 0.5, 0.5, 0.5, 0, "",
/*8-15*/ "","","","", 1, 1, 1, 0,
/*16-23*/ 0, 0, 0, 0, 0, 0, 0, 0,
/*data p1*/ 0, 8.5, 0.5, 1, 1.3, 0, 0, 0.5,
/*p2*/ 2.5, 10, 0.7, 2, 2.5, 0, 0, 0.8,
/*p3*/ 5, 7, 0.8, 0, 3.0, 0, 0, 0.5,
/*p4*/ 5, -8, 0.5, 0, 2.5, 0, 0, 0.5,
/*p5*/ 3, -10, 0.5, 0, 2, 0, 0, 0.5,
/*p6*/ 1.5, -9, 0.5, 0, 1.5, 0, 99, 0.5,
/*p7*/ 0, -7, 0.5, 0, 1.0, 0, 2, 0.5,
/*p8*/ 5, 0, 0.5, 0, 1.0, 0, 1, 0.5 }
```

Definiert wird die Schattierung hier im Offset 12 durch den Wert 1. Wie wir bereits bei den Vokalen gesehen haben, steht der Offset 12 für den Zeichentyp (token type). Bei Vokalen haben wir hier den Wert 2 (virtuelles Zeichen) gesetzt, für alle anderen Zeichen den Wert 0 (normales Zeichen). Der Wert 1 nun bedeutet für VSTENO, dass das Zeichen in jedem Fall schattiert werden soll (also unabhängig davon, welcher Vokal oder welches Zeichen vorausgeht).

Bitte beachten Sie in diesem Zeichen "AUCH" eine weitere Besonderheit: Da "AUCH" einem hoch gestellten "CH" entspricht, konnte hier einfach die Definition von "CH" kopiert und in den Offsets 13 und 14 (inconditional y before/after), die wir bereits gesehen haben, der Wert +1 gesetzt werden (was bedeutet, dass das ursprüngliche Zeichen "CH" einfach um eine Stufe nach oben verschoben wird). Definitionen dieser Art sind sehr effizient, da man bereits definierte Zeichen wiederverwenden (und wie hier für eine Kürzung gebrauchen) kann. Wir werden später noch weitere Möglichkeiten sehen, um aus bereits definierten Zeichen zusätzliche - kombinierte oder verschobene - Zeichen zu erstellen.

Das Kopieren eines Zeichens hat aber auch den Nachteil, dass die Definition u.U. nicht optimal ist: Im Falle von "AUCH" verwendet das urprüngliche Zeichen "CH" z.B. einen early exit point in Punkt P6 (Wert 99 an vorletzter Stelle im Datentupel). Da die Kürzung "AUCH" immer alleine steht, kann man den early exit point auch durch einen normalen Endpunkt ersetzen und die Punkte P7 und P8 löschen:

```
"AUCH" => { /*header0-7*/ 5, 1,-1, 0.5, 0.5, 0.5, 0, "",
/*8-15*/ "","","","", 1, 1, 1, 0,
/*16-23*/ 0, 0, 0, 0, 0, 0, 0, 0,
/*data p1*/ 0, 8.5, 0.5, 1, 1.3, 0, 0, 0.5,
/*p2*/ 2.5, 10, 0.7, 2, 2.5, 0, 0, 0.8,
```

falsch geschrieben, da sich bei "RR" die Schreibline in jedem Fall um eine Stufe erhöht!

```
/*p3*/ 5, 7, 0.8, 0, 3.0, 0, 0, 0.5,
/*p4*/ 5, -8, 0.5, 0, 2.5, 0, 0, 0.5,
/*p5*/ 3, -10, 0.5, 0, 2, 0, 0, 0.5,
/*p6*/ 1.5, -9, 0.5, 0, 1.5, 0, 1, 0.5 }
```

Eine letzte Bemerkung zur Kürzung "AUCH": Es ist VSTENO vollkommen egal, ob Sie nun eine Kürzung oder ein Zeichen definieren. VSTENO betrachtet alles, was gezeichnet werden kann, als Zeichen. Die Namen der Zeichen können einen oder beliebig viele Buchstaben lang sein (auch die Kürzung "VIELLEICHT" wird - obwohl es in der Langschrift 10 Buchstaben lang ist - von VSTENO als 1 Zeichen betrachtet). Auch das Aneinanderreihen von Buchstaben und/oder Kürzungen macht für VSTENO keinen Unterschied: Das Wort "dafür" zum Beispiel wird von VSTENO als zwei Kürzungen betrachtet, welche als zwei Zeichen (schattiertes D + normales F) aneinandergereiht wird. Wir werden später bei den Regeln sehen, wie Kürzungen - und die Übertragung derselben aus der Langschrift - definiert werden können.

Alternative exit points

Man würde nun vielleicht denken, dass wir langsam alle Besonderheiten stenografischer Zeichen abgedeckt haben, aber dem ist nicht so: Es gibt weitere Zeichen, die nach Spezialfunktionen verlangen und dazu gehören jene, welche Folgezeichen auf zwei verschiedene Arten anschliessen können: entweder (1) "normal": also auf der gleichen Schreiblinie wie bei 95% der Fälle) oder aber (2) "anders": in ganz wenigen Fällen. Zu diesen Zeichen gehört z.B. das vokalische R "VR", welches Folgezeichen normalerweise auf der Grundlinie anschliesst (vgl. "gern": die Zeichen r und n stehen auf der gleichen Linie), die Endungskürzungen "(D)EN" und "(D)EM" hingegen am oberen Ende anschliesst (vgl. "äusseren": das Zeichen r steht auf der Grundlinie, die Endung en hingegen eine halbe Stufe höher. Dies können wir mit dem Offset 16 alternative exit point (alternativer Endpunkt) im Header definieren[40]:

```
"VR" => { /*header0-7*/ 5, 0.5, 0, 0.5, 2, 2, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 2.5,
/*16-23*/ 5, 0, 0, 0, 0, 0, 0, 0,
/*data p1*/ 2.5, 5, 0.5, 1, 1.5, 0, 0, 0.5,
/*p2*/ 5, 2.5, 0.7, 0, 3.0, 0, 0, 0.7,
/*p3*/ 2.5, 0, 0.7, 0, 1.0, 0, 0, 0.5,
/*p4*/ 0, 2.5, 0.7, 0, 1.0, 0, 0, 0.5,
/*p5*/ 2.5, 5, 0.5, 0, 1.0, 0, 1, 0.0 }
```

Wie wir sehen können, enthält der Header im Offset 16 den Wert 5. Dies bedeutet nun, dass der "alternative Endpunkt" auf der y-Achse 5 Pixel höher liegen soll. Dies entspricht dem y-Wert des letzten Punktes P5 - also dem Kopfende des Zeichens "VR". Die Definition einer x-Koordinate ist nicht nötig, da diese automatisch beim Aneinanderfügen der Zeichen errechnet wird.

Die Frage ist nun: Wann kommt dieser alternative Endpunkt zum Einsatz. Hierzu sehen wir uns die Definition der Endung "EN" an:

[40] Der Einfachkeit halber werden in der Definition die intermediate shadow points weggelassen.

```
"EN" => { /*header0-7*/ 5, 0, 0, 0.5, 0, 0, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 1, 0, 0, 0, 0, 0, 0,
/*data*/ 5, 0, 0, 1, 1.0, 0, 0, 0,
/**/ 5, 0, 0, 0, 1.0, 0, 1, 0 }
```

Der entscheidende Wert steht hier im Offset 17 des Headers: Dieser Wert entspricht dem exit point to use (zu verwendender Endpunkt). VSTENO handhabt dies nun folgendermassen: Verlangt ein Zeichen - wie in diesem Fall die Kürzung "EN" - nach einem alternativen Endpunkt, so prüft VSTENO, ob das vorhergehende Zeichen einen solchen definiert. Ist dies der Fall (wie beim Zeichen "VR"), dann wird er verwendet. Bietet das vorhergehende Zeichen keinen alternativen Endpunkt an, so wird der normale Endpunkt verwendet (was das richtige Ergebnis liefert, vgl. z.B. "laufen": hier wird die Endung "EN" auf der Grundlinie angeschlossen).

Bietet ein Zeichen einen alternativen Endpunkt an und das Folgezeichen verlangt nicht danach, so wird der alternative Endpunkt ignoriert.

Absolute Koordinaten

Eine weitere Frage betrifft die Koordinaten: Wie werden diese von VSTENO gehandhabt? Prinzipiell können diese als absolute oder relative Werte betrachtet werden. Deshalb muss VSTENO angegeben werden, wie es die Koordinaten verrechnen soll. Dies geschieht im Offset 18, der für die "Interpretation y-Koordinate" steht. Das Feld kann zwei Werte annehmen:

Wert	Bedeutung
0	y-Koordinaten sind relativ (standard)
1	y-Koordinaten sind absolut

Diese Einstellung tut genau das, was der Name sagt: Sämtliche Zeichen, die wir bis jetzt definiert haben, verwenden relative Koordinaten (Standardeinstellung, Wert 0), was bedeutet, dass die Zeichen bei Höher- oder Tieferstellung automatisch "mitverschoben" werden. Wird hier der Wert 1 gesetzt, so wird die y-Koordinaten in den Punkten absolut gesetzt. Dies kann verwendet werden, wenn Zeichen unter keinen Umständen verschoben werden sollen[41].

Unverbundene Zeichen

Zeichen können verbunden (mit Vorzeichen) oder unverbunden sein. Dies trifft zum Beispiel auf Zahlen zu. Diese werden - auch in einem stenografischen Text - als normale Zahlen in Blockschrift geschrieben (und dürfen somit nicht an das vorangehende Zeichen angeschlossen werden). Als Beispiel hier die Definition der Zahl 1 (die wir wiederum deshalb wählen, weil sie nur aus zwei geraden Strichen besteht und damit sehr einfach ist):

[41] Wir fügen hier kein Beispiel an, weil das Phänomen in der Grundschrift kaum vorkommt. In der Eilschrift hingegen gibt es die Kürzung "Ding(e)", welche dem Zeichen "NG" in unschattierter Höherstellung entspricht. Dieses Zeichen muss unbedingt abgetrennt und in Höherstellung geschrieben werden.

```
"1" => { /*header0-7*/ 7, 0, 0, 0, 4, 0, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 0, 1, 0, 0, 0, 1, 0,
/*data p1*/ 0, 11, 0, 1, 1.0, 0, 0, 0,
/*p2*/ 7, 19, 0, 0, 1.0, 0, 0, 0,
/*p3*/ 7, 1, 0, 0, 1.0, 0, 1, 0 }
```

Der Wert 1 im Offset 22 bedeutet also "dieses Zeichen nicht verbinden" (der Wert 0 - den wir bis jetzt immer verwendet haben - bedeutet hingegen "dieses Zeichen verbinden").

Alternativ können Sie auch das Feld dr (draw) im Datentupel des ersten Punktes P1 auf den Wert 5 (= don't connect) setzen:

```
"1" => { /*header0-7*/ 7, 0, 0, 0, 4, 0, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 0, 1, 0, 0, 0, 0, 0,
/*data p1*/ 0, 11, 0, 1, 1.0, 5, 0, 0,
/*p2*/ 7, 19, 0, 0, 1.0, 0, 0, 0,
/*p3*/ 7, 1, 0, 0, 1.0, 0, 1, 0 }
```

Zu guter Letzt: Wenn Sie auf Nummer sicher gehen wollen, können Sie sowohl 1 im Header also auch 5 im draw-Feld des Punktes eintragen ... ;-)[42]

Anfangsspannung

Wie wir gesehen haben, müseen bei Bezier-Kurven für den Anfangs- und Endpunkt je zwei Spannungen (tensions) definiert werden: eine Eingans- und eine Ausgangsspannung. Die Spannungen stehen in den Felder t1 und t2 der 8er-Datentupel:

```
x1, y1, t1, d1, th, dr, d2, t2
```

Hierbei bezeichnet t1 die Ausggangsspannung des Punktes und t2 die Eingangsspannung des folgenden Punktes. Logischerweise fehlt bei dieser Art der Datenspeicherung die Eingangsspannung des ersten Punktes. Um diesen Mangel zu beheben wird deshalb die Eingangsspannung des allerersten Punktes in den Offset 3 des Headers geschrieben.

[42] Zu bevorzugen ist in diesem Fall die Variante "Header": Es handelt sich hier ja um eine Information, die das ganze Zeichen betrifft. Das draw-Feld des Punktes hingegen sollte verwendet werden, wenn INNERHALB eines Zeichens gewisse Punkte nicht miteinander verbunden werden. WICHTIGER HINWEIS: Mit der SE1 rev1 wurde das dr-Feld neu definiert: Grundsätzlich wurde der Integer-Wert innerhalb des 8er-Datentupels umgedeutet. Der alte dr-Wert der SE1 rev0 entspricht nun den bits 0-3. Die höheren Bits (4-15) werden dazu verwendet, Informationen bzgl. orthogonal und proportional Knots (sowie deren dazugehörigen parallelen Rotationsachsen) unterzubringen. Diese Lösung war deshalb nötig, weil innerhalb des 8er-Tupels nirgendwo mehr Platz war, um weitere Informationen unterzubringen. Da die Rückwärtskompatibilität aufgrund der Komplexität der SE1 (in der zusätzlichen Interaktion mit VPAINT) nur teilweise erreicht wurde, bleiben diese Funktionen in der Version 0.1rc deaktiviert (sie werden hier somit nur der Vollständigkeit halber erwähnt). Eine genaue Dokumentation der Verwendung der einzelnen dr-Bits findet sich als Kommentar im File https://github.com/marcelmaci/vsteno/blob/master/php/se1_backports.php.

Die Standardregel für diesen Wert lautet: Zeichen, die mit einer Rundung beginnen (z.B. "M"), sollten hier den Wert 0.5 (oder ähnlich) enthalten, Zeichen die spitz beginnen (z.B. "B" oder "T"), den Wert 0.

Zeichenteile

Ursprunglich konnten in der SE1 nur einzelne Zeichen definiert werden. Es kann jedoch auch Sinn machen, Zeichen als eine Folge (Aneinanderreihung) von einzelnen (und für verschiedene Zeichen verwendbare) Teile zu definieren. Dies wird erreicht, indem im Header im Offset 12 (Zeichentyp) der Wert 4 gesetzt wird: Die nachfolgenden Daten werden dann als "Zeichenteile" betrachtet. Zeichenteile unterscheiden sich von Zeichen dadurch, dass sie miteinander verbunden (joined) werden können.

Nehmen wir z.B. das Zeichen SP in Stolze-Schrey: Wir könnten dieses als Zeichen mit drei Einzelteilen betrachten: SP1 = runder Anfangsbogen, SP2 = gerader Strich (direkt auf den Bogen folgend) und SP3 = runder Endbogen (direkt auf den geraden Strich folgend). Wenn wir diese Bestandteile nun als "Teile" (Wert 4 in Offset 12) definieren, so wird VSTENO sie nahtlos aneinanderfügen. Das heisst zum Beispiel, dass der Endpunkt von SP1 und der Anfangspunkt von SP2 nur einmal eingefügt wird (da sie zusammenfallen, genau gleich bei SP2 und SP3). Der verbundene (joined) Punkt zwischen SP1 und SP2 erhält die Eingangsspannung von SP1 und die Ausgangsspannung von SP2.

Der Punkttyp wurde eingefügt in der Annahme, dass sich Zeichenverbindungen einfacher und präziser modellieren lassen (indem der Parser z.B. genau weiss, wie ein Zeichen endet oder anfängt, da Anfänge und Enden als separate Zeichenteile definiert sind). Allerdings verkompliziert dies den Parser und es zeigte sich auch, dass die Zeichenabstände durch Kategorisierung der Zeichen bereits relativ präzis definiert werden können.

Die Funktion, Zeichenteile aneinanderfügen zu können, kann dennoch für den einen oder anderen Fall nützlich sein und wird drum in der SE1 rev0 belassen.

Stenofont

Nachdem wir nun gesehen haben, wie man ein Zeichen definiert, stellt sich die Frage: Wohin mit diesen Informationen, damit VSTENO sich auch verarbeiten kann? Betrachten wir noch einmal die komplette Definition des Zeichens "T"[43]:

```
"T" => { /*h*/ 0, 0.5, 0, 0, 0, 0, 0, "",
/**/ "", "", "", "", 0, 0, 0, 0,
/**/ 0, 0, 0, 0, 0, 0, 0, 0,
/*d*/ 0, 20, 0, 1, 3, 0, 0, 0,
/**/ 0, 0, 0, 0, 1, 0, 1, 0,
/**/ 0, 2.5, 0, 4, 1, 0, 0, 0.5 }
```

[43] Sie weicht von den obigen leicht ab: z.B. verwendet sie als Vor/Nachbreite des Zeichens den Wert 0 (da der Abstand zwischen den Zeichen später von der Funktion "Spacer" eingefügt wird)

Diese Definition ist somit eines von vielen Zeichen, die wir zu einem so genannten Font (Schriftart) zusammenfassen. Ein Font wiederum ist Teil eines so genannten Modells (das einem Stenografischen System entspricht). Ein Modell enthält nebst dem Font (Zeichendefinitionen) auch Regeln (rules), die angeben, wie die Zeichen verwendet werden sollen, sowie einen Header (Kopf), der allgemeine Informationen zum Modell enthält[44].

VSTENO liest die Angaben für ein Modell - also das Font und die Regeln - aus der Datenbank. Wenn Sie ein eigenes Font (Modell) erstellen wollen, benötigen Sie deshalb als erstes ein Benutzer/innen-Konto. Loggen Sie sich mit diesem Konto ein und wählen Sie das Modell custom, indem Sie links unten auf den Button standard klicken. Zur Erinnerung: Das Modell custom ist jener Datenbank-Eintrag, den Sie für Ihr persönliches Modell verwenden können (das Modell Standard hingegen kann nicht überschrieben werden).

Wählen Sie nun, wenn Sie das Modell custom gewählt haben, unter dem Punkt Edit (nur sichtbar, wenn Sie eingeloggt sind) jenen Bereich des Modells aus, den Sie editieren wollen:

- Header: Enthält allgemeine Informationen zum Modell.
- Zeichen: Enhält das Font, bestehend aus Zeichen (base), kombinierten Zeichen (combiner) und verschobener Zeichen (shifter).
- Regeln: Enhält die Rules, also die Regeln mit denen VSTENO einen Langsschrifttext in Kurzschrift umschreibt.

Wenn Sie auf einen dieser Bereich klicken, dann öffnet VSTENO einen Texteditor mit den Informationen (Definitionen) aus dem entsprechenden Bereich. Wichtig ist nun, dass diese Bereiche in Sections und SubSections eingeteilt sind, die mit den Schlüsselwörtern #BeginSection() - #EndSection() und #BeginSubSection() - #EndSubSection() abgetrennt sind. Innerhalb der Klammern wird angegeben, um welchen Teil des sich handelt:

- Die Sections entsprechen den oben genannten Teilen, also: #BeginSection(header) - #EndSection(header); #BeginSection(font) - #EndSection(font); #BeginSection(rules) - #EndSection(rules)
- Die SubSections wiederum bedeutet:
  - Innerhalb der Section font: den Teilen base, combiner und shifter, die durch #BeginSubSection(base) - #EndSubSection(base); #BeginSubSection(combiner) - #EndSubSection(combiner); #BeginSubSection(shifter) - #EndSubSection(shifter) abgegrenzt sind.
  - Innerhalb der Section rules: jeweils einer sogenannten function (Funktion), also Regelteilen, die als logische Einheiten betrachtet und abgerbeitet werden (z.B. #BeginSubSection() - #EndSubSection(bundler); für jene Regeln, die mehrere Zeichen zu einem Zeichen bündeln).

[44] In der SE1 rev0 ist der Header prinzipiell leer, kann aber dazu verwendet werden, Kommentare zum Modell zu machen.

Am besten, Sie werfen einen Blick auf das Modell der Grundschrift Stolze-Schrey[45]. Sie sehen dort, wie die Subsections abgetrennt werden.

Die Zeichendefinitionen, die wir bis jetzt angeschaut haben, gehören somit in die Section font, und innerhalb der Section font in die Subsection base (da es sich um Basisdefinitionen handelt, also Zeichen, die als Grundzeichen definiert werden).

Wir werden nun sehen, wie diese Grundzeichen mithilfe des Combiners und Shifters verwendet werden können, um weitere kombinierte und verschobene Zeichen zu generieren.

Kombinierte Zeichen

Nachdem wir nun die Struktur eines Modells in den grossen Zügen kennen (und wissen, wohin wir die Definitionen speichern müssen, damit sie von VSTENO verwendet werden können), widmen wir uns noch einmal den Zeichen (also dem Font), um ein paar weitere, sehr hilfreiche Funktionen zu erläutern.

Im System Stolze-Schrey gibt es bekanntlich Zeichen, welche sich mit anderen "kombinieren" können. Es sind dies vor allem R und L in Verbindung mit verschiedenen Konsonantenzeichen wie z.B. T, B etc. Wir könnten nun für diese Kombinationen einfach neue Zeichen definieren. D.h. in unserer Gesamtliste an Zeichen würden wir zuerst ein Zeichen "T" definieren, dann ein Zeichen "T+R" und schliesslich ein Zeichen "T+L". Das Problem hierbei: Wir betreiben damit doppelt und dreifachen Aufwand! Es wäre viel effizienter, wenn VSTENO eine Funktion anböte, um gewisse Zeichen automatisch zu kombinieren - und eine solche Funktion existiert in der Tat: Sie nennt sich TokenCombiner.

Damit der TokenCombiner zwei Zeichen verbinden kann, muss er jedoch wissen, wo (d.h. an welcher Stelle) Zeichen miteinander verbunden werden können und auf welche Zeichen dies angewandt werden soll. VSTENO stellt hier die Funktion so genannter connection points (Verbindungspunkte) zur Verfügung: Ein Verbindungspunkt ist ein Punkt eines Zeichens, der im Datentupel-Feld d1 (= Offset 3) den Wert 4 enthält. Als Beispiel zeigen wir wieder unser Zeichen "T":

```
"T" => { /*header*/ 0, 0.5, 0, 0,4,2.5, 0, "",
/**/ "","","","", 0, 0, 0, 0,
/**/  0, 0, 0, 0, 0, 0, 0, 0,
/*data p1*/ 0, 20, 0, 1, 3.0, 0, 0, 0,
/*p2*/ 0, 0, 0, 0, 1.0, 0, 1, 0,
/*p3*/ 0, 2.5, 0, 4, 1.0, 0, 0, 0.5 }
```

Wir haben hier das Zeichen "T" also um einen zusätzlichen dritten Punkt P3 ergänzt, welcher die Koordinanten (0,2.5) aufweist. Dieser Punkt liegt also auf halber Höhe eines halbstufigen Zeichens - und ist somit der ideale Ansatzpunkt, um ein "R" einzufügen.

Das Problem ist hier jedoch, dass wir nicht die bereits definierten r (d.h. "VR" oder "AR") verwenden können, da diese entweder oben bzw. bei 12 Uhr ("VR") oder unten bzw. bei 6 Uhr ("AR") beginnen. Deshalb definieren wir hier ein spezielles

[45] https://github.com/marcelmaci/vsteno/blob/master/ling/grundschrift_stolze_schrey_redesign.txt

R, welches bei 3 Uhr - also rechts auf Viertelhöhe, d.h. genau bei der Koordinate (0,2.5) - beginnt:

```
"@R" => { /*header0-7*/ 5, 0.5, 0, 0.5, 0, 1, 0, "",
/*8-15*/ "","","","", 0, 0, 0, 0,
/*16-23*/ 0, 0, 0, 0, 0, 0, 0, 0,
/*data p1*/ 0, 0, 0.7, 0, 1.0, 0, 0, 0.7,
/*p2*/ -2, 2, 0.7, 0, 1.0, 0, 0, 0.5,
/*p3*/ -4, 0, 0.7, 0, 1.0, 0, 0, 0.5,
/*p4*/ -2, -2, 0.5, 0, 1.0, 0, 2, 0.5,
/*p5*/ 0, 0, 0.5, 0, 1.0, 0, 1, 0.7 }
```

Beachten Sie nun, dass wir dieses spezielle R - das wir "@R" nennen, nun nicht an der Koordinate (0,2.5) - wo das Zeichen letztlich eigentlich hinkommen soll -, sondern an der Ursprungskoordinate (0,0) beginnen. Mit anderen Worten: Sämtliche Koordinaten dieses "Verbundzeichens" sind als relativ zu verstehen. Beim Einfügen werden hier also jeweils die Koordinaten des connection point - im dem Falle also (0,2.5) - dazuaddiert. Die Punkte P1-P5 definieren einen Kreis im Gegenuhrzeigersinn, der im Ursprung endet (wo er begonnen hat). Beachten Sie auch, dass der Punkt P4 als Drehpunkt definiert wird: Dadurch wird sichergestellt, dass Zeichen, die tiefergestellt werden (wie z.B. in "Thron") später elegant angeschlossen werden.

Nachdem wir unsere beiden Einzel-Zeichen nun definiert haben, brauchen wir VSTENO nur noch zu sagen, dass wir die Zeichen kombinieren wollen. Wir tun dies, indem wir dem TokenCombiner vier Informationen übergeben:

```
“T” => { “@R”, 0, 0 }
```

Die 4 Informationen, die der TokenCombiner benötigt sind somit: (1) erstes Zeichen: "T", (2) zweites Zeichen: "@R", (3) Vorgängiges Delta-y: 0, (4) Nachträgliches Delta-y: 0.

Diese Zeile genügt also, um das neue Zeichen "T@R" zu generieren. Beachten Sie hierbei, dass der Name des kombinierten Zeichens immer die Zusammenfügung von "Zeichen1" + "Zeichen2" ist (hier also "T@R"). Die Werte vorgängiges und nachträgliches Delta-y werden in den Header des neuen Zeichens geschrieben. Genauer gesagt: Der TokenCombiner verwendet den Header des ersten Zeichens als Basis und ersetzt dann nur diese beiden Felder durch die angegebenen Werte. Ebenfalls passt der TokenCombiner die Breite des neuen Zeichens automatisch an (die Breite des neuen Zeichens entspricht der Summe der beiden Zeichenbreiten). Zu guter Letzt: Wenn wir zwei Zeichen kombinieren, besteht natürlich die Gefahr, dass das neue Zeichen mehrere Anfangs- und Endpunkte hat. Der TokenCombiner analysiert deshalb das neue Zeichen, löscht überflüssige Anfangs- oder Endpunkte (indem er sie in "normale" Punkte umwandelt) und behält nur den ersten und letzten Punkt des Zeichens als Anfangs- und Endpunkt.

Der TokenCombiner ist ein ungemein praktisches Werkzeug, um im Handumdrehen neue Zeichenkombinationen zu generieren. Z.B. genügt es, beim Zeichen "D" einen Verbindungspunkt P3 = 0, 2.5, 0, 4, 1.0, 0, 0, 0.5 wie im Zeichen "T" einzusetzen und den TokenCombiner zu ergänzen:

In #SubSection(base):

```
        "D" => { /*header*/ 0, 0.5, 0, 0, 2 ,3, 0, "",
        /**/ "","","","",0,0,0,0,
        /*+++*/ 0,0,0,0,0,0,0,0,
        /*data*/ 0, 10, 0, 1, 3.0, 0, 0, 0,
        /**/ 0, 0, 0, 0, 1.0, 0, 1, 0,
        /*connection point*/ 0, 2.5, 0, 4, 1.0, 0, 0, 0.5 }   // neue Zeile
```

In #SubSection(combiner):

```
"T" => {"@R", 0, 0 }
"D" => {"@R", 0, 0 }
```

Zwei Zeilen genügen somit, um das neue Zeichen "D@R" zu generieren. Da dieses - wie die übrigen Grundzeichen - ins Hauptfont von VSTENO geschrieben wird, können Sie es danach ohne Einschränkung wie jedes andere Zeichen verwenden!

Punktschlingen

Der TokenCombiner funktioniert in den meisten Fällen recht gut. Allerdings zeigte sich, dass bei der Verbindung mit sehr kleinen Zeichen - in Stolze-Schrey zum Beispiel - das Punktschlingen-L das Problem entstehen kann, dass das zweite Zeichen aufgrund der Schattierung des ersten Zeichens kaum sichtbar ist. Mit anderen Worten: Die Schattierung ist so breit, dass sie einen Teil des kombinierten Zeichens überdeckt.

Als Lösung empfahl es sich hier, den TokenCombiner um einen so genannten Randvektor (border vector) zu ergänzen. Der Randvektor ist jener Wert der angibt, wo bei einem schattierten Zeichen der Aussenrand der Schreiblinie verläuft. Zur Illustration zeigen wir hier die Kombination des Zeichens "P"

```
"P" => {
   /*header*/ 4, 0.5, 0, 0, 0, 0, 0, "",
         /**/ "", "", 0.0, 0.75, 0, 0, 0, 0,
         /**/ 0, 0, 0, 0, 0, 0, 0, 0,
   /*data*/ 0, 20, 0, 1, 3, 0, 0, 0,
       /**/ 0, 2.5, 0.5, 0, 2.5, 0, 0, 0.5,
       /**/ 2, 0, 0.5, 4, 1.5, 0, 0, 0.5,    // Verbindungspunkt
       /**/ 2, 0, 0.5, 0, 1.5, 0, 2, 0.5,    // Weiterführungspunkt
       /**/ 4, 2, 0, 0, 1, 0, 1, 0
}
```

mit dem Punktschlingen-L "@L"

```
"@L" => {
    /*header*/ 0, 0, 0, 0.5, 2, 2, 0, "",
          /**/ "", "", "", "", 0, 0, 0, 0,
          /**/ 0, 0, 0, 0, 0, 0, 0, 0,
    /*data*/ 0, 0, 1, 0.5, 1, 0, 0, 0.5,
```

```
            /**/ 1, 1, 0.5, 0, 1, 0, 0, 0.5,
            /**/ 0, 2, 0.5, 0, 1, 0, 0, 0.5,
            /**/ -1, 1, 0.5, 0, 1, 0, 2, 0.5,
            /**/ 0, 0, 0.5, 0, 1, 0, 1, 0.5
    }
```

die wie gewohnt über den TokenCombiner verbunden werden:

```
    #BeginSubSection(combiner)
        "B" => { "@L", 0, 0 }
    #EndSubSection(combiner)
```

Der entscheidende Unterschied liegt hier in den Offsets 10 und 11 des Headers: Die Werte bvectx = 0.0 und bvecty = 0.75 entsprechen hier dem besagten Randvektor R(0.0/0.75). Was aber bedeuten diese Werte genau?

Wie wir sehen liegt der Verbindungspunkt (Wert 4 im Offset 3 des Datentupels) im Punkt V(2,0), was dem Fusspunkt des Zeichens B entspricht. Normalerweise würde nun genau hier (d.h. mit diesen Koordinaten) das zweite Zeichen mit dem ersten verbunden. Allerdings beträgt die Dicke des Zeichens bei Schattierung an dieser Stelle 1.5. Dieser Wert ist relativ (relative_thickness) und gibt die Strichdicke im Vergleich zur Grunddicke an, die im Formular unter Dicke bzw. mit der Session-Variable 'token_thickness' angegeben werden. Ausserdem kann die Schattierung des Zeichens ihrerseits mit dem Wert Schattierung im Formular bzw. der Session-Variablen 'token_shadow' weiter differenziert werden. Die endgültige Strichdicke D berechnet sich also wie folgt:

```
    D = relative_thickness * token_size * token_shadow
```

Der genaue Wert von D braucht uns aber nicht zu kümmern. Uns interessiert hier nur der relative Wert (relative_thickness) und 1.5 bedeutet in dem Fall "eineinhalb mal dicker" als die Grunddicke (ganz egal, welche zusätzlichen Werte der/die Benutzer/in hier wählt). Da das von uns definierte Punktschlingen-L rund 3 Einheiten hoch ist (von y=+2 bis y=-1) wird es folglich zur Hälfte durch die Schattierung des Grundzeichens verdeckt.

Der Randvektor R(0.0/0.75) bedeutet nun, dass wir dies korrigieren, indem wir die Koordinaten des zweiten Zeichens "@L" relativ um 0.75 auf der y-Achse nach oben verschieben (die Modellierung erfolgt hier bei 90 Grad, also am aufrecht stehenden Zeichen). Dadurch wird also nur noch ein Viertel (0.75 von 3) durch die Schattierung verdeckt, wodurch das Zeichen besser lesbar wird.

Allerdings ergibt sich ein Problem: Das Grundzeichen "P" wurde so definiert, dass zuerst der Verbindungspunkt (2,0), danach das Verbundzeichen "@L" und schliesslich ein Weiterführungspunkt (2,0) eingefügt wird. Da der Verbindungspunkt vom TokenCombiner durch den Eintrittspunkt des Verbundzeichens ersetzt wird, ergäbe sich hier die Situation, dass dieser nicht - wie benötigt - bei (2,0) zu liegen käme, sondern bei (2,0.75) - eben um den Vektor R(0.0,0.75) nach oben verschoben. Dadurch würde das Grundzeichen B "verbogen". Um dies zu verhindern, wird der Eintrittspunkt des Verbundzeichens nicht relativ verschoben. Auch dies hat negative

Effekte: Da nun der Eintrittspunkt - im Unterschied zu allen anderen Punkten des Verbundzeichens - nicht verschoben wird, stimmen nun die Relationen der Punkte im Folgezeichen untereinander nicht mehr ganz (und somit wird das Folgezeichen verbogen). Dieser Fehler ist innerhalb des Verbundzeichens jedoch kaum sichtbar und stellt somit eine akzeptable und pragmatische Lösung dar, um die Sichtbarkeit solch kleiner Zeichen zu verbessern.

Verschobene Zeichen

Ähnlich wie der TokenCombiner funktioniert auch der TokenShifter: Dieser ermöglicht es, Zeichen horizontal oder vertikal zu verschieben. Im System Stolze-Schrey kann dies z.B. dazu verwendet werden, Anschlüsse von Zeichen an ein Aufstrich-T zu definieren, ohne das entsprechende Zeichen noch einmal neu eingeben zu müssen. Wir tragen in die #Subsection(shifter) also folgende Zeile ein:

```
"N" => { "&TN", 4, 15, 0, 1.5 }
```

Der TokenShifter benötigt 6 Informationen, die folgendes bedeuten: (1) zu verschiebendes Zeichen: "N", (2) Name des neuen Zeichens: "&TN", (3) Verschiebung auf x-Achse: 4 Pixel nach rechts, (4) Verschiebung auf y-Achse: 15 Pixel (= 1.5 Stufen) nach oben, (5) vorgängiges Delta-y: 0 (Schreibzeile verschiebt sich nicht), (6) nachträgliches Delta-y: 15 (Schreibzeile verschiebt sich um 15 Pixel bzw. 1.5 Stufen nach oben). Das Grundzeichen "N" wird durch diese Anweisung also um 1.5 Stufen nach oben und 4 Pixel nach rechts verschoben. Das neue Zeichen heisst "&TN" und die Schreiblinie verschiebt sich nach dem Zeichen um 1.5 nach oben. Wenn nun dieses Zeichen im Wort "Zentner" vom Grundlinien-N aus verbunden wird, so entspricht "&TN" einem Aufstrich-T. Die neue Schreiblinie befindet sich am Fusspunkt des Zeichens "N".

Auch dies ist eine sehr effiziente Art, neue Zeichen zu generieren. Für das neue Zeichen "&TENS", welches einem Aufstricht-T verbunden mit dem Grundzeichen "NS" entspricht, genügt z.B. die folgende zusätzliche Zeile in #SubSection(shifter):

```
"N" => { "&TN", 4, 15, 0, 1.5 }
"NS" => { "&TENS", 4, 10, 0, 1 }
```

Soll statt dem Aufstrich-T die Kürzung "HEIT" verwendet werden, genügt eine grössere Verschiebung auf der x-Achse:

```
"N" => { "&TN", 4, 15, 0, 1.5 }
"NS" => { "&TENS", 4, 10, 0, 1 }
"NS" =>{ "&EITENS", 18, 10, 0, 1 }
```

Beachten Sie dass die vertikale Verschiebung hier nur 10 Pixel beträgt, da das Zeichen "NS" im Unterschied zu "N" einstufig ist. Aus diesem Grund kommt auch die neue Schreiblinie nur 1 Stufe höher zu liegen (also eine halbe Stufe tiefer als bei "N")[46].

[46] Abschliessend ist zu erwähnen, dass dies nur eine Möglichkeit ist, das Aufstrich-T mit anschliessendem Zeichen zu definieren. Wenn die Stenozeichen nur noch mit einer Nettobreite (also

Diakritische Zeichen

Diakritische Zeichen sind Zeichen, welche Wörtern, einzelnen Zeichen oder Vokalen hinzugefügt werden, um Zweifelsfälle (also zum Beispiel gleichlautende oder gleichgeschriebene) Wörter oder Silben (insbesondere zum Beispiel Präfixe) mit mehrfacher Bedeutung voneinander zu unterscheiden. Beispiele in Stolze-Schrey sind: das Aufheben von Kürzungen (kleiner, schräger Strich unterhalb von b oder g zur Aufhebung der Vorsilben be- und ge-), Unterscheidung Gross/Kleinschreibung (flacher Strich unterhalb b bedeutet Grossschreibung), Unterscheidung ai/ei (Punkt oberhalb der weiten, hochgestellten, nicht-schattierten Verbindung bedeutet ai), Unterscheidung lange/kurze Vokale (u-förmiges Böglein oder Strich oberhalb der Vokalverbindung).

Solche diakritische Zeichen lassen sich hinsichtlich des Schreibvorgangs dadurch charakterisieren, dass der Stift jeweils kurz abgesetzt wird, um diese Zeichen unter- oder oberhalb einzufügen. Da sich Stenografie prinzipiell dadurch auszeichnet, dass - in Hinblick auf eine höhere Schreibgeschwindigkeit - der Stift nicht abgesetzt wird, müssen Sie innerhalb von VSTENO speziell umgesetzt werden. Als Beispiel: Ein/e menschliche/r Stenograf/in hat zwei Möglichkeiten, Stenogramme mit diakritischen Zeichen zu verfassen: (1) Er/Sie schreibt das Wort bis zu jener Stelle, wo ein diakritisches Zeichen eingefügt werden muss, unterbricht dann kurz, fügt das Zeichen ein und schreibt anschliessend das Wort zu Ende; (2) Er/Sie schreibt zuerst das Ganze Wort zu Ende und fügt erst am Schluss das diakritische Zeichen dort ein, wo es benötigt wird. Der menschliche Geist ist in dem Sinne bemerkenswert, als dass es ihm leicht fällt, diese beiden Methoden beliebig zu verwenden:

- Im ersten Fall (Absetzen) fällt es ihm leicht, den Stift nach dem Einfügen des diakritischen Zeichens wieder dort anzusetzen, wo er den Schreibfluss unterbrochen hat und den Rest des Stenogrammes - inklusive eventueller "Bögen", die begonnen wurden - zu Ende zu führen.

- Im zweiten Fall (Einfügen am Schluss) fällt es ihm leicht, im Nachhinein jene genaue Stelle im Stenogramm zu finden, wo das diakritische Zeichen hingehört.

Computer verfügen nicht über eine derartige "Intuition" und somit muss VSTENO sehr präzise Anweisungen erhalten, wie diakritische Zeichen eingefügt werden sollen. Ähnlich wie im Fall (2) schreibt VSTENO zuerst das gesamte Stenogramm (als

ohne Vor/Nachbreite) definiert werden, kann auf die Kreierung eines weiteren Zeichens via Shifter vollständig verzichtet werden. Stattdessen wir einfach ein Aufstrich-T definiert, welches nur den Punkt (18,10) als Eingangs- und Endpunkt enhält. Aufgrund der "Nettobreite" der Zeichen, wird das Folgezeichen nun unmittelbar angeschlossen. Man muss dann einzig noch darauf achten, dass die Schreibzeilenverschiebung richtig gehandhabt wird (kann z.B. direkt über verschiedene Definitionen des Aufstrich-T mit verschiedenen Schreiblinien erreicht werden). Dies ist inzwischen auch das Schöne an VSTENO: Die SE1 rev0 bietet hinsichtlich des Parsers (Regeln) und der Zeichendefinitionen z.T. mehrere Möglichkeiten, um die eine oder andere spezifische Eigenschaft eines Stenosystems umzusetzen. Gleiches gilt hier z.B. für die Kürzung "AUCH": wir haben diese weiter oben mithilfe der Header-Felder inconditional_y_before realisiert. Es wäre aber u.U. auch möglich gewesen, hier den Shifter zu verwenden (einziges Problem das hier zusätzlich zu betrachten ist, wäre hier die Endschlaufe).

eine durchgehende Folge von Bezierkurven, die in einem Spline definiert sind) und fügt dann - in einer zweiten (elektronischen) "Handbewegung" gewissermassen - ein diakritisches Zeichen ein, indem es einen zweiten, separaten Spline zeichnet (der wiederum eine durchgehende Linie bzw. eine Folge von Bezier-Kurven darstellt).

Wie aber "merkt" sich (bzw. "weiss") VSTENO an welcher Stelle ein diakritisches Zeichen eingefügt werden muss? Ganz einfach: Diakritische Zeichen werden als kombinierte Zeichen gehandhabt, d.h. es kann wiederum der TokenCombiner verwendet werden, um ein diakritisches Zeichen mit einem Basiszeichen zu verbinden. Hier ein Beispiel für das Zeichen "B": wir wollem diesem einen diakritischen Schrägstrich unterhalb einfügen, um die Vorsilbe be- nach Bedarf aufgheben zu können. Hierfür definieren wir zunächst unser Grundzeichen:

```
"B" => {
  /*header*/ 4.5, 0.5, 0, 0, 0, 0, 0, "",
        /**/ "", "", "", "", 0, 0, 0, 0,
        /**/ 0, 0, 0, 0, 0, 0, 0, 0,
  /*data*/ 0, 10, 0, 1, 3, 0, 0, 0.5,
      /**/ 0, 3, 0.5, 0, 2.5, 0, 0, 0.5,
      /**/ 2.5, 0, 0.5, 4, 1.5, 0, 0, 0.5,
      /**/ 2.25, 0, 0.5, 0, 1.5, 0, 2, 0.5,
      /**/ 4.5, 2, 0.5, 0, 1, 0, 1, 0,
      /**/ 2, -5, 0, "@#/", 0, 0, 0, 0
}
```

Wir erkennen hier das letzte Datentupel { 2, -5, 0, "@#/", 0, 0, 0, 0 }, welches an vierter Stelle den Wert "@#/" enthält. Wir erinnern uns: Die Information an vierter Stelle (= Offset 3, = Datenfeld 1, = d1) bezeichnet den Eintritts-Punkttyp (mit möglichen Werten wie 1=entry knot, 2=pivot point oder 4=kombiniertes Zeichen). Der Wert "@#/" gibt nun an, dass sich das Basiszeichen B mit dem diakritischen Zeichen "@#/" verbinden kann.

Diese diakritische Zeichen "@#/" definieren wir nun ebenfalls:

```
"@#/" => {
  /*header*/ 4.5, 0.5, 0, 0, 0, 0, 0, "",
        /**/ "", "", "", "", 0, 0, 0, 0,
        /**/ 0, 0, 0, 0, 0, 0, 0, 0,
  /*data*/ 0, 0, 0.0, 1, 1.0, 0, 0, 0,
      /**/ -5, -4, 0.0, 0, 1.0, 0, 1, 0.0
}
```

Wie wir sehen enthält es im Datenteil (data) nur 2 Punkte: P1(0,0) und P2(-5/-4). Beide Punkte haben ausserdem einen Eintritts- und Austrittsspannung von 0.0 (was einer spitzen - bzw. hier: geraden - Verbindung entspricht). Dies ist das, was wir wollen: eine gerade Linie von links unten P2(-5,-4) nach rechts oben P1(0,0)[47].

[47] In der Beschreibung "von links unten nach rechts oben" wurde natürlich die Reihenfolge der Punkte P1 und P2 vertauscht. VSTENO ist es egal, in welche Richtung die Linie gezeichnet wird (in diesem Fall also von rechts oben nach links unten - was wohl am ehesten der natürlichen Ausführung eines/r menschlichen Stenografen/in entspricht.

Wo aber steht nun dieser Schrägstrich? Oder anders gesagt: Wie werden die Koordinaten der Punkte P1 und P2 interpretiert? Da sich diakritische Zeichen mit verschiedenen Zeichen verbinden können (und folgedessen unter Umständen auch an verschiedenen Positionen eingefügt werden müssen) verwendet VSTENO hier eine relative Koordinatenangabe. Die Punkte P1(0,0) und P2(-5,-4) werden ausgehend vom Ursprungspunkt aus berechnet, die in der Definition von "B" angegeben sind: Grundpunkt G(2/-5). Die absoluten Koordinaten ergeben in diesem Fall P1'(2,-5) und P2'(-3,-9). Der Grundpunkt gibt also an, in welchem Abstand zum Basiszeichen das diakritische Zeichen steht, in unserem Fall also G(2,-5) was bedeutet, dass der Schrägstrich halbstufig unterhalb und etwa in der Mitte des Grundzeichens "B" beginnen soll.

Bitte beachten Sie bei der Definition diakritischer Zeichen zwei Dinge: (1) diakritische Zeichen werden nicht geneigt, sondern mit den Koordinaten eingefügt, die Sie angeben (diakritische Zeichen sehen somit unabhängig von der Neigung immer gleich aus); (2) Der Grundpunkt hingegen wird mit dem Grundzeichen mitgeneigt, d.h. diakritische Zeichen unterhalb der Grundlinie verschieben sich z.B. bei einer Zeichenneigung von 60 Grad nach links, diakritische Zeichen oberhalb der Grundlinie nach rechts (je höher umso weiter nach rechts).

Um das Grundzeichen und das diakritische Zeichen miteinander zu verbinden, verwenden wir schliesslich den TokenCombiner:

```
#BeginSubSection(combiner)
    "B" => { "@#/", "", "" }
#EndSubSection(combiner)
```

Entscheidend ist hier, dass wir an zweiter und dritter Stelle, dass wir einen leeren String "" verwenden: VSTENO erkennt daran, dass es sich nicht um eine Zeichenverbindung mithilfe eines Verbindungspunktes (connection point, Typ 4) handelt, sondern um ein Zeichen, das als separaten Spline gezeichnet werden soll[48]. Das neue entstandene, kombinierte Zeichen lautet auch hier: Grundzeichen + diakritisches Zeichen, also "B@#/"[49].

Regeln

Nach den Zeichendefinitionen kommen wir nun zu den so genannten Regeln, welche einzelne Wörter der Langschrift Schritt um Schritt so umwandeln, dass sie am Schluss durch Aneinanderfügen der definierten Zeichen als Stenogramme dargestellt werden können. Diese "Übertragung" findet in mehreren Schritten statt und um diese

[48] Zur Erinnerung: Die "klassische" Form des TokenCombiners lautet "B" => { "@R6", 0, 0 }. Hier wird das Zeichen "B" mit "@R6" (= R mit Fusspunkt bei 6 Uhr) verbunden. Die Zahlen-Werte 0 und 0 stehen hier für die zusätzliche Zeichenbreite vor und nach dem kombinierten Zeichen stehen.

[49] Der TokenCombiner ordnet das Zeichen "B@#/" derselben Spacer-Gruppe zu wie das Grundzeichen. Durch die Verwendung diakritischer Zeichen wird die Abstandsberechnung noch einmal komplizierter, da weitere Zeichen entstehen, die die möglichen Kombinationen (Permutationen) weiter in die Höhe treiben.

sichtbar zu machen, können Sie in der Demoversion die Funktion "Debug" anwählen. Für das Wort "baten" sehen Sie dann[50]:

```
ORIGINAL: baten
[1] WORD: bat{EN} FROM: rule: (?<!(^[Ww])|i)en$ => {EN}
[2] WORD: bAt{EN} FROM: rule: a => A
[3] WORD: BAt{EN} FROM: rule: b => B
[4] WORD: BAT{EN} FROM: rule: t => T
NUMBER OF RULES APPLIED: 4
```

Beim Wort "baten" wird also zuerst die Endkürzung -en erkannt und durch {EN} markiert (Schritt 1), danach wird der Vokal a erkannt und mit dem Grossbuchstaben A markiert (Schritt 2). Analog dazu werden auch die Konsonanten b und t erkannt und mit den Grossbuchstaben B und T markiert (Schritte 3+4). Die so entstanden Folge - die VSTENO intern als TokenList (Zeichenliste) bezeichnet - kann danach sehr simpel zu einem Stenogramm verarbeitet werden, indem die Zeichendefinitionen aus der Variable $steno_tokens_master ausgelesen und die einzelnen Zeichen aneinandergefügt werden.

Das obige Beispiel ist natürlich relativ simpel, da alle Zeichen ohne grosse Veränderungen verwendet werden können. Gewisse Zeichen müssen aber je nach Kontext anders geschrieben werden. So z.B. das Aufstrich-T, wie es in "bunt" vorkommt:

```
ORIGINAL: bunt
[1] WORD: bun[&T] FROM: rule: ([bcdfghjklmnpqrvwxyz])t => $1[&T]
[2] WORD: bUn[&T] FROM: rule: u => U
[3] WORD: BUn[&T] FROM: rule: b => B
[4] WORD: BUN[&T] FROM: rule: n => N
NUMBER OF RULES APPLIED: 4
```

Alle diese Dinge müssen VSTENO anhand von Regeln genauestens "erklärt"[51] bzw. beigebracht werden. Damit VSTENO die Anweisungen versteht, müssen müssen sie in einer klar definierten Formelsprache abgefasst werden. Im Falle von VSTENO ist dies REGEX, eine Formelsprache die standardmässig in PHP integriert ist. Regex ist ein sehr mächtiges Instrument, das einige Tücken aufweist ... das aber - richtig angewandt - sämtlichen linguistischen Bedürfnissen gerecht werden kann.

Beachten Sie bitte, dass Regeln nach dem Format "Wenn A, dann B" funktionieren. In den obigen Beispielen bedeutet "b => B" also: "Wenn du innerhalb des Wortes den Kleinbuchstaben b findest, dann erstetze ihn durch den Grossbuchstaben B".

[50] Die Angaben können leicht von der aktuellen Version abweichen, da das Modell für Stolze-Schrey ständig angepasst wird. Wichtig ist hier einfach das Prinzip, dass VSTENO mehrere (z.T. recht viele) Regeln sequentiell anwendet, um ein Stenogramm zu generieren. Ebenfalls wichtig: Die Debug-Funktion! Nutzen Sie diese unbedingt, wenn Sie eigene Stenografie-Systeme erstellen wollen. Sie hilft Ihnen dabei herauszufinden, welche Regeln angewandt wurden und welche nicht, und Sie können damit Fehler in Ihren Regeln - und zum Beispiel in der Abarbeitung derselben - leichter finden!

[51] Wenn man dieses Verb im Zusammenhang mit einem Computer verwenden kann ... ;-) Aber wir haben VSTENO weiter oben ja in der Tat als unseren digitalen Stenoschüler definiert und in diesem Sinne ist die Analogie triftig: Der Computer bzw. das Programm VSTENO ist unser Discipulus, dem wir als Linguisten Stenografie beibringen (und "erklären").

REGEX

Die Möglichkeiten von REGEX (Abkürzung für so genannte "regular expressions") auch nur ansatzweise darzustellen, würde den Rahmen dieses Tutorial sprengen - schliesslich gibt es ganze Bücher, die sich ausschliesslich mit REGEX beschäftigen! Wir werden uns also damit begnügnen, hier nur einige wesentliche Elemente zu erklären. Für den Rest verweisen wir Sie auf die folgenden Seiten, die Ihnen weiterhelfen können:

Ein guter Start, um einen Überblick zu REGEX zu erhalten, sind die beiden Wikipedia-Seiten auf Deutsch und auf Englisch:

- Deutsch: https://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck
- Englisch: https://en.wikipedia.org/wiki/Regular_expression

Ebenfalls sehr zu empfehlen ist der folgende REGEX-Tester:

- Online-REGEX-Tester: https://regex101.com/

Er erlaubt es Ihnen, einzelne REGEX-Ausdrücke direkt im Webbrowser zu testen und so Fehler in bzw. falsch forumlierte Formeln zu finden und zu korrigieren.

REGEX kann man durchaus als "learning by doing" lernen: Spielen Sie also mit den Elementen die wir im Folgenden vorstellen im REGEX-Tester etwas herum und Sie werden bald ein relativ intuitives Verständnis dafür erlangen, wie REGEX funktioniert!

Wortgrenzen und Lookaround-Expressions

Hier nun also einige wenige Grundregeln und Prinzipien von REGEX, die Sie für linguistische Regeln mit Sicherheit benötigen werden. Beginnen wir mit den Zeichen ^ und $. Sie markieren den Anfang und das Ende eines Wortes:

```
"^hab$" => "{HAB}"
```

Bedeutet also: Falls das Wort genau der Zeichenfolge "hab" entspricht (bzw. sich vor "hab" der Wortanfang und nach "hab" das Wortende befindet), dann ersetze die Zeichenfolge "hab" durch "{HAB}". Beachten Sie, dass die Ausdrücke ^hab$ und {HAB} innerhalb von Anführungszeichen "" gesetzt werden müssen, da es sich dabei um Strings (Zeichenfolgen) handelt. Diese Regel kann also verwendet werden, um die Kürzung HAB zu definieren. Sie ist allerdings schlecht formuliert: Sie berücksichtigt z.B. nicht, dass das Verb "haben" auch am Wortanfang stehen kann (was bedeutet, dass "hab" dann mit einem Grossbuchstaben beginnt). Ausserdem wird die Kürzung HAB auch in längeren Wörtern angewendet (wie z.B. "Inhaber", "habt", "haben" etc.). Auch diese Fälle würden nicht berücksichtigt, da die Regel verlangt, dass das Wort nach "hab" zu Ende ist. Wir formulieren die Regel deshalb um:

“[Hh]ab” => “{HAB}”[52]

Die eckige Klammer [] bedeutet hier: “einer der aufgelisteten Buchstaben” (die Liste kann beliebig lang sein - z.B. [aeiou] für die 5 Grundvokale - oder man kann auch weitere Ausdrücke verwenden wie z.B. [a-z] oder [A-Z] für Kleinbuchstaben und Grossbuchstaben, [0-9] für Ziffern etc.). Da wir die Zeichen ^ und $ (Wortanfang und Wortende) entfernt haben, trifft diese Regel nun auf die oben genannten Fälle zu: “Inhaber”, “habt” und “haben” werden zum Beispiel in “In{HAB}er”, “{HAB}t” und “{HAB}en” umgewandelt. Allerdings ist auch diese Regel zu ungenau formuliert, da auch Wörter wie “schaben”, “Haber”[53] in “sc{HAB}en” und “{HAB}er” umgewandelt werden, was wiederum falsch ist ...

Dies können wir korrigieren, indem wir eine so genannte Lookaround-Expression (zu Deutsch etwa: Schau-dich-um-Ausdruck) verwenden. Es gibt zwei Arten von Lookaround-Expressions: die Lookbehind-Expression (sie sucht nach Zeichenfolgen vor dem Wort) und die Lookahead-Expression (für Zeichenfolgen nach dem Wort). Ausserdem können Lookaround-Expressions positiv (= die Zeichenfolge muss vorkommen) oder negativ (= die Zeichenfolge darf nicht vorkommen) sein. Im Folgenden verwenden wir eine negative Lookbehind-Expression. Diese steht in einer runden Klammer vor dem Wort und beginnt mit ?<!:

“(?<![Ss]c)[Hh]ab” => “{HAB}”

Diese Regel bedeutet also: Ersetze “hab” - egal ob mit Gross- oder Kleinbuchstaben beginnend - an einer beliebigen Stelle des Wortes durch {HAB}, sofern nicht “sc” oder “Sc” vorausgeht. Oder anders formuliert: Ersetze “hab” nur dann, wenn das H nicht Teil von SCH ist. Diese Regel löst also das Problem von “schaben”, nicht aber jenes von “Haber”. Wir könnten natürlich auch hier eine negative Lookahead-Expression verwenden:

“[Hh]ab(?=!er)” => “{HAB}”

Allerdings würde dann das Wort “Inhaber” ebenfalls nicht mehr in “In{HAB}er” umgewandelt.

Man sieht also: Das Formulieren von präzisen linguitischen Regeln - die wirklich nur auf jene Wörter angewandt wird, auf die die Regel wirklich zutrifft - ist zum Teil recht anspruchsvoll. Es lohnt sich aber, hier Zeit zu investieren, denn je präziser die Regel ist, umso weniger “Ausnahmen” müssen später zusätzlich (in einem Wörterbuch beispielsweise) definiert werden.

[52] Beachten Sie bitte gleich hier, dass “[Hh]ab” => “{HAB}” und “\[Hh\]ab” => “{HAB}” nicht dasselbe ist. “[Hh]ab” bedeutet “H oder h”, es könnte in REGEX auch “(H|h)ab” geschrieben werden. Das “escapte” “\[Hh\]ab” hingegen würde bedeuten: das Zeichen [und] muss vorhanden sein (würde also auf “[Hh]ab” zutreffen). Kurzum: REGEX ist ebenso effizient wie heimtückisch; kleine Zeichen - besonders wenn es ums Escaping mit \ geht - können eine ganz andere Bedeutung haben ... Ebenfalls zu beachten ist die Tatsache, dass in PHP Zeichen nur auf der IF-Seite escaped werden müssen!

[53] Wir meinen hier das umganssprachliche Wort für “Hafer”.

Klammern, Quantoren und Variablen

Quantoren sind Zeichen, welche angeben, wie oft ein Zeichen vorkommen muss. REGEX kennt hier z.B. die Zeichen ? (= 0 oder 1 Mal vorkommend), + (= mindestens 1 Mal vorkommen), * beliebig oft vorkommend. Diese können auf einzelne Zeichen angewandt werden oder es können mehrere Zeichen zu einer Gruppe zusammengefasst werden, für die der Quantor gilt. Als Beispiel betrachten wir folgende Regel:

```
“^([Uu]n)?[Zz]uver” => “$1{ZU}{VER}”
```

Hier wurde also der Ausdruck [Uu]n durch die Klammern () zu einer Gruppe zusammengefasst. Der Quantor ? nach ([Uu]n) bedeutet, dass die Vorsilbe un- (die am Anfang des Wortes - markiert durch das Zeichen ^ - gross oder klein geschrieben werden kann) 0 oder 1 Mal vorkommen kann. Dieses “Muster” (oder pattern, wie es auf Englisch genannt wird) trifft z.B. auf Wörter wie “unzuverlässig”, “Unzuverlässigkeit” zu (hier kommt un- 1 Mal vor), aber auch auf Wörter wie “zuverlässig”, “Zuversichtlichkeit” usw.

Auf der rechten Seite (der Folge-Seite der Regel) steht der Ausdruck $1 für die “Variable in der ersten Position”. Mit Position ist gemeint: “Der x-te Klammerausdruck von links beginnend”, in unserem Fall also ([Uu]n).

Diese Regel nimmt somit folgende Ersetzungen vor: “unzuverlässig” => “un{ZU}{VER}lässig”, “Unzuverlässigkeit” => “Un{ZU}{VER}lässigkeit”, “Zuversicht” => {ZU}{VER}sicht, “zuverlässig” => “{ZU}{VER}lässig. Beachten Sie hierbei, dass die Variable $1 den Klammerausdruck exakt so wiedergibt, wie er im Wort vorgefunden wird, als “Un” (mit grossem Anfangsbuchstaben) im Substantiv “Unzuverlässigkeit” und “un” (mit Kleinbuchstaben) im Adjektiv “unzuverlässig”. Nicht so hingegen beim Ausdruck “[Zz]u” (wo wir keine Variable verwenden): Hier wird also sowohl “Zu” und “zu” durch “{ZU}” ersetzt.

Da Variablen nummeriert sind, kann man auch mehrere hintereinander verwenden:

```
“^([Uu]n)?([Zz]u)ver” => “$1$2{VER}”
```

In dieser Regel z.B. wird nur das Präfix ver- durch {VER} ersetzt. Wie bei un- weiter oben wird nun auch in diesem Beispiel die zweite Vorsilbe “zu” exakt so übertragen, wie sie im Wort steht (also “Zu” bei einem Substantiv, “zu” bei klein geschriebenen Wörtern).

Im Unterschied zu ? bezeichnet der Quantor + einen Ausdruck der mindestens 1 Mal (oder mehr) vorkommen muss. Die Regel

```
“^([Zz]u)+” => “$1|”
```

Trennt eine oder mehrere Vorsilben zu- durch einen vertikalen Strich vom Rest des Wortes ab: “zugeben” => “zu|geben”, “zuzugeben” => “zuzu|geben”. Bitte beachten Sie, dass dies wiederum eine linguistisch zu unpräzis formulierte Regel ist, da sie auch auf Wörter wie “Zuzug” zutrifft (hier findet die Regel 2 Vorsilben zu-, dabei enthält das Wort nur 1 Vorsilbe - der zweite Teil gehört zum Stamm des Wortes).

Auch trifft die Regel theoretisch auf inexistente Wörter zu wie "zuzuzukumi" (hier fände die Regel 3 Vorsilben).

Wildcards und Greediness

REGEX definiert den Punkt . als so genannte Wildcard[54]: Dieses Zeichen kann also für "irgend ein" Zeichen stehen. Auch den Punkt können wir mit Quantoren kombinieren, wobei hier ein neuer Aspekt ins Spiel kommt: die so genannte greediness (zu Deutsch in etwa: Gefrässigkeit). Quantoren können also "greedy" (gefrässig) oder non-greedy (ungefrässig oder genügsam) sein. Im ersten Fall versucht REGEX den Ausdruck auf die grösstmögliche Zeichenfolge anzuwenden, im letzten Fall hingegen wird der kürzeste Ausdruck gesucht, der auf das Muster zutrifft.

```
[1] Bett(.*)en => Bett$1{EN}
[2] Bett(.*?)en => Bett$1{EN}
```

Im Wort "Bettenkapazitäten" (welches 2 x die Endung -en enthält), findet die erste Regel (mit dem "gefrässigen" Ausdruck *) die LETZTE Endung und ersetzt das Wort somit durch "Bettenkapazität{EN}". In der zweiten Regel (mit dem "genügsamen" Ausdruck *?) findet REGEX hingegen der ERSTEN Ausdruck, sodass das Resultat hier "Bett{EN}kapazitäten" lautet.

Passen Sie deshalb mit Quantoren besonders auf: Sie können eine total andere Bedeutung haben als man - von einer intuitiven, sprachlichen Logik her denkend - glauben würde ...

Logisches Oder

Wir haben bereits die eckigen Klammern [] kennengelernt, welche eine Reihe von Zeichen zusammenfasst, von denen wenigtens eines zutreffen muss. In der Logik entspricht dieses einem Oder: Im Ausdruck [abcd] muss also entweder a oder b oder c oder d vorkommen. Ein weiterer Ausdruck um ein logisches Oder wiederzugeben ist der vertikale Strich:

```
"(ck|kk)" => "[CK]"
```

Diese Regel bedeutet also, dass "Zucker" zu "Zu[CK]er" und "Mokka" zu "Mo[CK]a" wird. Beachten Sie in diesem Zusammenhang unbedingt, dass die eckige Klammer auf der linken Seite (Bedingung) der Regel nicht die gleiche Bedeutung hat wie auf der rechten Seite (Folge):

```
"[Ää]" => "[Ä]"
"(Ä|ä)" => "[Ä]"
```

[54] Wir auch Joker genannt.

Diese beiden Regeln sind gleichbedeutend![55] Die eckige Klammer auf der linken Seite bedeutet also wie erwähnt "eines dieser Zeichen" (also Ä oder ä) auf der rechten Seite bedeutet es jedoch "ersetzen durch das Klammerzeichen" (offen oder geschlossen). Diese Regel transformiert die Wörter "Kläger" und "KlÄger" zu "Kl[Ä]ger"!

Wenn Sie auf der linken Seite nach einer eckigen Klammer suchen möchten, dann müssen Sie das Zeichen "escapen" (wir übersetzen mal frei: "seiner REGEX-Bedeutung entheben"). Dies geschieht indem der Backslash \ vorangestellt wird:

```
"\[Ää\]" => "{Ää}"
```

Diese Regel würde die Zeichenfolge "ABC[Ää]XYZ" zu "ABC{Ää}XYZ" umschreiben!

Auch hier: Passen Sie bei Zeichen, die in REGEX eine spezielle Bedeutung haben, sehr gut auf, ob Sie sie wörtlich (als Literal oder Buchstabe) oder in der Bedeutung von Regex verwenden wollen. Dies trifft prinzipiell auf alle Zeichen zu, die in REGEX eine spezielle Bedeutung haben. Sicherheitshalber listen wir die wichtigsten im Folgenden noch einmal auf.

Escaping

Folgende Zeichen MÜSSEN in PHP-Regex "escaped" werden, wenn Sie "wörtlich" (= als genau dieses Zeichen) verwendet werden sollen:

```
. als \.
^ als \^
$ als \$
* als \*
+ als \+
? als \?
( als \(
) als \)
[ als \[
\ als \\
| als \|
```

Die geschwungenen Klammern {} KÖNNEN in PHP-REGEX "escaped" werden (oder nicht)[56]. Speziell weisen wir auch auf das Zeichen ^ hin. Es hat innerhalb der eckigen Klammern die Bedeutung von "nicht" (logische Negation):

```
"^[Zz]u([^m])" => "{ZU}$1"
```

[55] ... theoretisch. Aus irgendeinem unerfindlichen Grund - vermutlich hat es mit dem Sonderzeichen Ä zu tun - funktioniert, zumindest in meiner Systemkonfiguration, nur die Variante (Ä|ä). Dasselbe gilt auch für Üü, Öö, ÄU ... Ich empfehle deshalb, für diese Sonderzeichen, die Variante in runden Klammern zu verwenden.

[56] Ich empfehle sie nicht zu escapen, da die Regeln dadurch etwas übersichtlicher werden.

Dieses Regelbeispiel bedeutet also: Ersetze die Vorsilbe zu- am Anfang eines Wortes durch {ZU}, sofern das darauffolgende Zeichen kein m ist (in Stolze-Schrey wird in diesem Fall keine Kürzung verwendet). Beachten Sie, dass hier die beiden Zeichen ^ verschiedene Bedeutungen haben: Das erste bedeutet "Wortanfang", das zweite bedeutet "nicht m".

Wenn Sie das Zeichen ^wortwörtlich (also als Zeichen) suchen möchten, müssen Sie es "escapen":

```
"ABC\^XYZ" => "ABC{circonflex}XYZ"
```

Wandelt die Zeichenkette "ABC^XYZ" in "ABC{circonflex}XYZ" um!

Es kann nicht genug betont werden, wie wichtig diese unscheinbaren Unterschiede sind: REGEX ist deshalb so "kompliziert", "tricky", "kryptisch" - oder wie immer Sie es bezeichnen möchten - weil ein einziges Zeichen je nach Art und Weise, wie es verwendet wird, eine total andere Bedeutung haben kann! Verwenden Sie in diesem Fall den erwähnten REGEX-Tester und vergewissern Sie sich, dass Ihre Regel auch wirklich das heisst, was Sie meinen ...

Regeln in VSTENO

Die Regeln, die wir uns bis jetzt angesehen haben, entsprachen dem Standard-Schema der REGEX-Replace-Funktion, nämlich: Wenn A, dann (ersetze durch) B. VSTENO erweitert den Regelformalismus um eine weitere Möglichkeit: Wenn A, dann B, ausser C (oder D):

```
A => B                          normale REGEX-Regel
A => { B, C, D ..., X }         erweiterte Regel
```

In VSTENO können diese Formeln genau in dieser Weise notiert werden. Wir verwenden noch einmal die Kürzung "HAB", die wir bereits weiter oben gesehen haben:

```
"(?<![Ss]c)[Hh]ab" => { "{HAB}", "^Haber" }
```

Diese Regel bedeutet also: Ersetz die Zeichenfolge "hab" (oder "Hab") an einer beliebigen Stelle des Wortes durch "{HAB}", AUSSER das Muster "Haber" (mit Grossbuchstaben) steht am Anfang des Wortes. Diese Regel nimmt in den folgenden Beispielwörtern folgende Transformationen vor:

```
Inhaber => In{HAB}er
haben => {HAB}en
habt => {HAB}t
schaben => schaben
Haber => Haber
Habermacher => Habermacher
```

Beachten Sie, dass das zweite Element des Folgearrays "^Haber" von VSTENO ebenfalls als REGEX-Muster interpretiert wird, weshalb es auch auf "Habermacher" zutrifft (sonst müsste dort "^Haber$" stehen).

Auch diese Kürzungsregel für "HAB" ist nicht perfekt, da z.B. das Wort "Habicht" ebenfalls zu "{HAB}icht" gekürzt wird. Mit dem erweiterten Regelformalismus von VSTENO können wir weitere Ausnahmen aber sehr einfach im Folge-Array hinzufügen:

```
"(?<![Ss]c)[Hh]ab" => { "{HAB}", "^Haber", "Habicht" }
```

Auch hier hat die REGEX-Notation wieder den Vorteil, dass sämtliche Fälle (z.B. Genetiv "Habichts") erfasst werden.

Gross-/Kleinbuchstaben

VSTENO bietet eine weitere, sehr nützliche Erweiterung des Standard-REGEX-Regelformalismus, um Gross- in Kleinbuchstaben umzuwandeln (und umgekehrt):

```
"([A-Z])" => "strtolower()";
"([a-z])" => "strtoupper()";
```

Hier wird also mit ([A-Z]) und ([a-z]) nach Gross- und Kleinbuchstaben gesucht. Anschliessend wird die gefundene Zeichenkette mit strtolower() udn strtoupper(); in Klein- und Grossbuchstaben umgewandelt. Beachten Sie bitte die runde Klammer (die obligatorisch ist): Sie markiert jenen Teil des Wortes, der umgewandelt werden soll.

Beachten Sie bitte, dass die obige Regel die Umlaute (ä, ö, ü) oder andere Zeichen (z.B. mit Zirkumflex) ausser Acht lässt!

VSTENO bzw. die Regeln für Stolze-Schrey verwendet diese Umwandlungen mehrmals: Wenn man z.B. das Übertragen konsequent mit Kleinbuchstaben beginnt und alle Umwandlungen mit Grossbuchstaben vornimmt, dann weiss man genau, welche Teile des Wortes bereits umgeschrieben sind (alle Grossbuchstaben) und welche noch umgeschrieben werden müssen (alle Kleinbuchstaben).

Funktionen

Mit dem Debug-Modus konnten wir bereits einen Eindruck gewinnen, wie (in welcher Reihenfolge) VSTENO die Regeln innerhalb der Section Rules abarbeitet. Die Standardreihenfolge ist im Prinzip von oben nach unten. Aber es gibt auch Möglichkeiten, Gruppen von Regeln zu so genannten Funktionen zusammenzufassen und die Abarbeitungsreihenfolge dieser Funktionen selber zu definieren bzw. von Bedingungen abhängig zu machen. Aber der Reihe nach - betrachten wir zunächst ein Beispiel:

```
#BeginSubSection(relancer)
     "^(.*?)$" => "strtolower()"; // all to low
 "({.*?})" => "strtoupper()";    // {..} to upper again

     "(\[.*?\])" => "strtoupper()";  // [..] to upper again
#EndSubSection(relancer)
```

Dies Sequenz fasst 3 Regeln zur Funktion "relancer" zusammen: (1) alle Zeichen werden zu Kleinbuchstaben umgewandelt, (2) + (3) Kürzungen {..} und gebündelte Zeichen [..] werden wieder zu Grossbuchstaben umgewandelt. Beispiel: {VER}LO[VR]{EN} wird zu: {VER}lo[VR]{EN}[57].

Stages

Stages wurden als neues[58] Konzept eingeführt, um zu definieren, auf welchen Teil des Textes oder Wortes sich die Regeln beziehen:

```
stage 0: gesamter Text
stage 1: Wörterbuch / linguistische Analyse (einzelnes Wort)[59]
stage 2: ganzes Wort
stage 3: Teilwörter (in zusammengesetzten Wörtern)
stage 4: ganzes Wort
```

Betrachten wir einen Beispieltest:

```
Dies ist ein Beispieltext. Er enthält ganze "Sätze" und
einzelne "Wörter", sowie Satzzeichen (Punkte und Kommas)
und weitere Zeichen (zum Beispiel Leerzeichen, Klammern
und Anführungszeichen). D.h. also er enthält auch
diakritische Zeichen.
```

In Stage 0 "sieht" nun VSTENO den ganzen Text. D.h. es ist möglich, REGEX-Regeln zu schreiben, die auf den gesamten Text und sämtliche Zeichen zugreifen können:

```
"zum Beispiel" => "z.B.";
"\(" => "[";
"D\.h\." => "dh";
```

Hier werden also Leerzeichen zwischen zwei Wörtern entfernt und der Ausdruck gekürtz (Beispiel 1: "z.B."), runde durch eckige Klammern ersetzt (Beispiel 2) und Punkte aus einer Abkürzung entfernt (Beispiel 3).

In Stage 1 trennt VSTENO dann den Text in einzelne Wörter, wobei "Wort" bedeutet: alles, was zwischen zwei Leerzeichen steht. Zusätzlich werden alle Sonderzeichen (Punkte, Kommas, Klammern, Anführungszeichen etc.) herausgefiltert. Für die Regeln "sichtbar" sind nunmehr nur noch so genannte "Rohwörter" (bare words):

[57] Der relancer macht sich hier das vorhin erklärte Prinzip zunutze und markiert Wortteile, die nicht mehr umgeschrieben werden müssen als Grossbuchstaben, und alles, was noch umgeschrieben werden muss, als Kleinbuchstaben.

[58] Erweiterung des Metaparsers im Februar 2019.

[59] Der linguistical analyzer wurde ebenfalls neu eingeführt und wird unter anderem hier erläutert: https://www.vsteno.ch/docs/gel_speiende_spiegel.pdf

```
Beispieltext
Kommas
D.h
```

Hier wurden am Ende des Wortes also ein Punkt ("Beispieltext") und eine Klammer ("Kommas") entfernt. Beachten Sie, dass bei "d.h." nur der letzte Punkt entfernt wird (der Punkt im Innern bleibt also bestehen, da VSTENO nur Sonderzeichen am Anfang und am Ende eines Wortes entfernt).

Diese Rohwörter (bare words) werden nun (in Stage 1) an die Datenbank gesandt. Findet VSTENO keinen Eintrag, geht das Wort anschliessend zur linguistischen Analyse (linguistical analyzer).

Die Stage 2 "sieht" dann genau das gleiche wie die Stage 1, also einzelne Wörter, mit dem Unterschied, dass die Rohwörter nun durch die linguistische Analyse mit zusätzlichen Informationen versehen wurden. Es sind dies Silbengrenzen (-) für Silben (z.B. "wei-te-re"), Morphemgrenzen (+) für Vorsilben (z.B. "ent+hält"), Wortgrenzen (|) für zusammengesetzte Wörter (z.B. "Bei+spiel|text", "Leer|zei-chen") und Morphemgrenzen (#) für Nachsilben (z.B. "heim#lich"). Wir sprechen hier von der LNG-Form: Diese kann definiert werden als "Rohwort, die linguistische Information enthält und unter Umständen aus mehreren zusammengesetzten Wörtern besteht".

Die Stage 3 kommt dann bei zusammengesetzten Wörtern zum Zug. Sie "sieht" - der Reihe nach - die einzelnen "Teilwörter der LNG-Form":

- "Bei+spiel|text": Regeln werden zuerst auf "Bei+spiel", danach auf "text" angewandt.
- "ent+hält": Die Regeln werden auf "ent+hält" angewandt (ist kein zusammengesetztes Wort).

Teilwörter werden hier also als in einzelne, eigenstängige Wörter aufgebrochen, die einen eigenen Wortanfang und ein Wortende aufweisen, welche wiederum von REGEX-Regeln genutzt werden können. Die Stage 3 ist damit insbesondere für die korrekte Anwendung von Kürzungen wichtig.

Zur Illustration hier nun einige konkrete Beispiele, wie die Stages genutzt werden können:

- Stage 0: In Stolze-Schrey werden Abkürzungen ohne Punkte geschrieben. Da Satzzeichen in den Stages 1 bis 4 unsichtbar sind, können solche Regeln nur in der Stage 0 (gesamter Text ist sichtbar) umgesetzt werden. Die Regel "[Dd]\.h\." => "dh" zum Beispiel entfernt hier beide Satzzeichen (Punkte).
- Stage 1: Die Stage 1 ist ein spezieller Schritt und der Datenbank (Wörterbuch) bzw. der linguistischen Analyse vorbehalten. Aber auch hier ist es möglich, REGEX-Regeln zu schreiben, die sich auf ein Rohwort (bare word) beziehen. D.h. VSTENO sieht hier einzelne Wörter ohne Satz- und Sonderzeichen: "Teetasse." aus dem Originaltext, wird somit zum Rohwort "Teetasse", das ans Wörterbuch (Datenbank) gesandt wird. Findet VSTENO einen Eintrag für das Rohwort, so wird die weitere Berechnung des Stenogramms mit dem gefundenen Eintrag fortgesetzt (dies kann entweder die LNG-, STD- oder

PRT-Form sein, siehe später). Wird kein Eintrag gefunden, wendet VSTENO anschliessend eine linguistische Analyse an (auf die ebenfalls später genauer eingegangen wird). In unserem Fall wird dabei "Teetasse" zu "Tee|tasse" (d.h. VSTENO erkennt, dass es sich um ein zusammengesetztes Wort mit zwei Teilen handelt).

- Stage 2: VSTENO sieht hier das gleiche wie in Stage 1, also das gesamte Wort "Tee|tasse". Der einzige Unterschied: Das Wort wurde nun im Wörterbuch nachgeschlagen oder linguistisch analysiert und enthält zusätzliche Informationen. In unserem Fall ist dies das Zeichen |, welche eine Wortgrenze angibt. Wir könnten nun in Stage 2 eine Regel der Form "Tee|tasse" => "Tee\\tasse" schreiben, um VSTENO dazu zu bringen, das Wort getrennt zu schreiben.

- Stage 3: Hier sieht VSTENO jedes Teilwort (bei zusammengesetzten Wörtern) einzeln. Nehmen wir z.B. das Wort "Vertragsvermittler". VSTENO wird hier zwei Teilwörter erkennen: "Vertrags" + "Vermittler". Wenn wir nun eine Regel der Form "^Ver" => "{VER}" in Stage 3 schreiben, so wird die Kürzung verauf BEIDE Vorsilben angewandt: "{VER}trags|{VER}mittler". Schreiben wir die gleiche Regel in Stage 2, so wird nur die erste (am Wortanfang wegen des REGEX-Zeichens ^) ersetzt: "{VER}trags|vermittler".

- Stage 4: Hier kehren wird zur gleichen Stufe wie in Stage 2 zurück, d.h. wir sehen wieder das ganze Wort, inklusive Wortgrenze |. Wir können dies nutzen, um z.B. Wörter zu trennen. Die Regel "\[&T\]\|" => "[&T]\\"[60] trennt zum Beispiel Teilwörter, die auf Aufstrich-t (= [&T]) enden, vom folgenden Wort ab.

Wie aber definiere ich, welche Regeln in welcher Stage angewandt werden sollen? Hierfür werden die #BeginSubSection() und #EndSubSection()-Statements mit dem Stage-Schlüsselwort #>stageX kombiniert. X steht dabei für die Nummer der Stage:

```
#BeginSubSection(shortener,#>stage3)
   // hier beginnen Kürzungsregeln
   // der Stage 3
   // sie werden auf jedes Teilwort
   // einzeln angewendet
   "^ver" => "{VER}";
#EndSubSection(shortener,#>stage4)
```

Das Beispiel zeigt die oben bereits erwähnte Kürzungsregel[61]. #BeginSubSection(shortener,#>stage3) bedeutet also "die folgende Regel wird in Stage 3 ausge-

[60] Beachten Sie, dass bestimmte Zeichen - wie [und \ - in REGEX "escaped" werden müssen: \[is also gleichbedeutend mit [und \\ bedeutet \.

[61] Bitte beachten Sie, dass das Beispiel - der Übersichtlichkeit halber - stark vereinfacht ist. Eine so formulierte Regel würde viele falsche Wörter generieren (z.B. würde "Vers" zu "{VER}s") und müsste präziser als "^[Vv]er\+" => "{VER}" geschrieben werden (d.h. die Kürzung wird nur angewandt, wenn ver- von der linguistischen Analyse als Vorsilbe - mit einer darauffolgenden Morphemgrenze (+) - erkannt wurde.

führt). Die Zeile #EndSubSection(shortener,#>stage4) bedeutet "hier endet die Stage 3 und die folgende Regel wird in Stage 4 ausgeführt".

Bitte beachten Sie folgende wichtige Regeln im Umgang mit Stages:

- Stages müssen der Reihe nach abgearbeitet werden (Sie können also nicht von Stage 0 zu Stage 4 springen und anschliessend wieder zu Stage 3 zurückkehren).
- Stages müssen nahtlos aufeinanderfolgen (d.h. wenn Stage 2 endet - z.B. mit #EndSubSection(accentizer,#>stage2) - so muss die folgende Zeile zwingend mit #BeginSubSection(shortener,#>stage3) beginnen).
- Die Wörterbuch-Variablen (STD und PRT) dürfen nicht in Stage 3 definiert werden, da sie sonst nicht das ganze Wort enthalten (weisen Sie sie deshalb in Stage 4 zu!)

Jede andere Verwendung der Stages für zu unvorhersehbaren Resultaten[62]!

Zeichenabstände

Eine grosse Herausforderung bei stenografischen Systemen stellt der Abstand zwischen den einzelnen Zeichen dar. Zu Beginn definierte VSTENO pro Zeichen fixe Abstände vor und nach den Zeichen ([63]). Aber anders als bei normalen Fonts, wo die Abstände zwischen zwei Zeichen immer gleich gross sind[64], können - bzw. müssen - Stenografiezeichen normal, hoch oder tief oder eng und weit verbunden werden. Ausserdem können Stenografiezeichen auch sehr hoch (z.B. tt, pp, ff) oder sehr klein (s, n oder Häkchen) sein. Zu guter Letzt gibt es Zeichen, die rund oder spitz enden oder beginnen, das heisst es gibt Verbindungstypen spitz-spitz, spitz-rund, rund-spitz, rund-rund - und auch die Rundung selber kann je nach Zeichen ganz anders sein (z.B. weite Rundung in Vorsilbe ent-, hingegen enge Rundung in b, n, m).

Kurzum, wenn man all diese Parameter zusammen nimmt, ergibt sich eine ganz stattliche Summe an Möglichkeiten[65]. Daraus folgt auch, dass es eigentlich unmöglich ist, die Abstandsproblematik innerhalb der Zeichendefinitionen zu lösen. Als Al-

[62] You've been warned ... ! :)

[63] Dies sind die Offsets 4 und 5 im Zeichenheader (also offs_additional_x_before und offs_additional_x_after), die als legacy-Variablen für dei SE1 weiterhin unterstützt werden, aber im Grunde obsolet sind.

[64] Bei so genannten "monospaced" Fonts ist der Abstand unabhängig von der Zeichenabfolge, also fix und gleich zwischen allen Zeichen. Bei proportionalen Fonts variieren zwar die Abstände je nach Zeichen, sind aber - spezifisch für die Zeichenkombination - grundsätzlich ebenfalls fix, es sei denn es kommen weitere typografische Feinheiten wie Kerning und Stegausgleich (z.B. in Verbindung mit Blocksatz) hinzu.

[65] Ich ging ursprünglich von etwa 50000 Fällen aus, aber die Sache ist weit schlimmer als angenommen: Das System Stolze-Schrey besteht zwar offiziell nur aus 43 Zeichen, aber viele Zeichen müssen in verschiedenen Ausführungen hinterlegt werden, was die Zahl also erhöht. Ausserdem können sich sehr viele Zeichen mit R und L zu Sekundärzeichen kombinieren (z.B. br, bl, cr, cl, dr, dl etc.), was die Zahl noch einmal signifikant erhöht. Meine Schätzung geht im Moment dahin, dass es rein theoretisch mehrere 100'000 Möglichkeiten gibt! Nicht alle dieser "theoretischen" Kombinationen müssen speziell definiert werden. Insbesondere stellen weite Abstände grundsätzlich

ternative bot sich an, das Ganze in den Parser zu transferieren, sprich die Abstände über REGEX-Regeln zu definieren. Aufgrund der schieren Anzahl an Möglichkeiten, ist es jedoch praktisch unmöglich, entsprechende Regeln von Hand zu definieren. Die beste Lösung scheint somit, die Zeichen zu Gruppieren und die Generierung entsprechender Regeln zu automatisieren. Mit dem spanischen Modell wird nun erstmals das folgende Verfahren getestet, das in drei Schritten stattfindet:

1. Als erstes werden die Zeichen in Gruppen eingeteilt.
2. Danach werden Kombinationen und entsprechende Abstände definiert.
3. Schliesslich werden daraus Abstandsregeln generiert (die - wie andere Regeln - in das Modell eingefügt werden können).

Im Folgenden nun die Beschreibung, wie die einzelnen Schritte vorgenommen werden.

Gruppierung

Um die Zeichen zu gruppieren wird der Offset 23 im Header der Zeichendefinitionen genutzt. Hier vier Beispiele:

```
"B" => { /*header*/ 5, 0.5, 0, 0, 0, 0, 0, 0,
0, 0, "", "", 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, "L3:L3A:R1:R1A", /*data*/ ... }
"D" => { /*header*/ 0, 0.5, 0, 0, 0, 0, 0, 0,
0, 0, "", "", 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, "L1:L1A:R1:R1A", /*data*/ ... }
"P" => { /*header*/ 5, 0.5, 0, 0, 0, 0, 0, 0,
0, 0, "", "", 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, "L3:L3A:R1:R1B", /*data*/ ... }
"T" => { /*header*/ 0, 0.5, 0, 0, 0, 0, 0, 0,
0, 0, "", "", 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, "L1:L1A:R1:R1B", /*data*/ ... }
```

Wichtig ist hier zuerst einmal, dass jedes Zeichen mehreren Gruppen zugeordnet werden kann (hierfür werden die einzelnen Gruppen durch Doppelpunkt : voneinander abgetrennt). In der obigen Gruppe ordnen wir die Zeichen B, P, D, T je 4 Gruppen zu, wobei wir zwischen Hauptgruppen, Hauptkategorien und Unterkategorien unterscheiden, die jeweils durch einen Buchstaben oder eine Zahl bezeichnet werden.

überhaupt kein Problem dar (da "ein bisschen weiter" oder "ein bisschen weniger weit" optisch wenig Unterschied macht). Aber wenn nur schon die engen Abstände in den Varianten hoch (Vokale au, i = 2 Möglichkeiten), normal (Vokal a oder kein Vokal = 2 Möglichkeiten) und tief (Vokale o, u = 2 Möglichkeiten) in Verbindung mit all den Sekundärzeichen definiert werden müssen, ergibt sich sich schnell eine 6-stellige Zahl an Verbindungen. Aktuell sind z.B. im spanischen System 128520 Kombinationen durch spezifische Regeln definiert!

Die Buchstaben L und R bedeuten linke und rechte Hauptgruppe, d.h. der entsprechende Fall tritt ein, wenn das Zeichen links (als erstes Zeichen) oder rechts (als zweites, nachfolgendes Zeichen) steht.

Die Zahlen bezeichnen dann die Gruppennummer bzw. die Zeichenkategorie. Die Zeichen B, D, P und TT werden zum Beispiel alle derselben rechten Gruppe R1 zugeordnet, weil alle spitz und mit einem vertikalen String beginnen (in diesem Sinne verhalten sie sich als Folgezeichen gleich).

Hingegen haben B, P und D, T verschiedene linke Gruppen: B und P enden rund auf der Grundlinie (Gruppe L3), D, T endet zwar auch auf der Grundlinie, ist aber spitz und ausserdem "schlanker" als das Zeichen B, deshalb Gruppe L1. D.h. diese zwei Gruppen (L1, L3) verhalten sich als linke Zeichen (d.h. wenn sie zuerst stehen) anders.

Schliesslich unterscheiden wir durch den Buchstaben A und B weitere Unterkategorien, in diesem Falle nämlich, ob die Zeichen ein- oder zweistufig sind: A bedeutet hier einstufig (hierzu gehören B und D, sowohl als linke Zeichen L1A und L3A als auch als rechte Zeichen R1A und R3A), B bedeutet zweistufig (hierzu gehören P und T, sowohl als linke Zeichen L1B und L3B als auch als rechte Zeichen R1B und R3B).

Wir können die Klassifizierung der Zeichen in Haupt- und Untergruppen beliebig weiterführen und z.B. PP und TT mit einer weiteren Bezeichnung C analog dreistufigen Gruppen L1C, L3C, R1C, R3C zuordnen.

Sämtliche Bezeichnungen können frei gewählt werden und beliebig lang sein. Es besteht auch kein Unterschied, ob Buchstaben oder Zahlen verwendet werden. Die obgenannten Fälle sind also nur Beispiele (und statt L1A könnte man die Gruppe grad so gut "linkeschlankeeinstufigezeichen" nennen). Die Empfehlung lautet hier einfach: möglichst kurze Zeichen verwenden (der Übersichtlichkeit halber) und Untergruppen sollten kein überschneidenden Mengen bilden[66].

Die Zuordnung zu den Gruppen wird vom PARSER selbständig vorgenommen. Sämtliche Zeichen werden intern der Variable $token_groups zugeordnet. Es wird also nicht unterschieden, ob ein Zeichen zum Beispiel ein Vokal oder ein Konsonant ist. Aus diesem Grund sollte die automatische Zuweisung nur für Konsonanten verwendet werden. Für "Zwischenzeichen" (also Zeichen, die zwischen zwei anderen Zeichen vorkommen können, im System Stolze-Schrey typischerweise Vokal und Diphtonge) sollten manuell durch das Setzen der SESSION-Variable $vowel_groups im Header-Teil der Modell-Definition definiert werden, also innerhalb der Schlüsselwörter #BeginSubSection(session) und #EndSubSection(session)[67]:

```
#BeginSubSection(session)
    "spacer_vowel_groups" := "
        V1:[A,O,U],
        V2:[I,AU]";
#EndSubSection(session)
```

[66] Wenn die Hauptgruppe R1 = { D, B, T, P, TT, PP } ist, dann sind Untergruppen R1 = { R1A = { D, B }, R1B ={ T, P }, R1C = { TT, PP } } korrekt. Falsch wäre R1 = { R1A = { D, B, T }, R1B ={ T, P, D }, R1C = { TT, PP } }, da sich R1A und R1B überschneiden.

[67] Das genaue Format wird im folgenden Abschnitt "Kombinationen" erklärt.

Hier werden zwei Vokalgruppen (V1, V2) definiert, welche zwischen normal und tiefen (a, o, u) und hochgestellten (i, au) Vokalen unterscheiden. Wir definieren hier nur zwei Gruppen, d.h. wir gehen davon aus, dass zum Beispiel die Abstände in den Kombinationen dd, dad, dod, dud gleich gross sind[68]. Speziell definieren müssen wir sicher die Hochstellung, da hier gerade beim Aufeinandertreffen von breiten linken und hohen rechten Zeichen mehr Abstand benötigt wird (ein Beispiel ist hier "stritt": das ST in Kombination mit dem R und der Kopfschlaufe kommt dem "Aufstrich" Richtung TT in die Quere, weshalb es etwas weiter auseinander geschrieben werden muss als zum Beispiel die Verbindung d-i-d in Didier).

Kombinationen

Sobald die Zeichen in Gruppen und Unterguppen eingeteilt sind, können Zeichenkombinationen definiert werden. Dies kann durch Setzen von SESSION-Variablen im Header-Teil der Modell-Definition erreicht werden, also innerhalb der Schlüsselwörter #BeginSubSection(session) und #EndSubSection(session):

```
#BeginSubSection(session)
    "spacer_token_combinations" := "
        C1:[L1,R1],
        C2:[L3,R1A],
#EndSubSection(session)
```

Nach dem Format "variable" := "wert"; kann hier also Variablen spacer_token_combinations ein Wert zugewiesen werden. Der Wert hat das Format:

```
X1:[Y1,Z1],X2:[Y2,Z2],...,Xn:[Yn,Zn]
```
[69]

X entspricht der Bezeichnung der Zeichenkombination (also welchen "Namen" wir ihr geben), Y und Z verweisen auf die zuvor definierten Zeichengruppen.

Im obigen Beispiel definieren wir 2 Zeichenkombinationen: die Kombination C1, die aus einem ersten Zeichen aus der Gruppe L1 und einem Folgezeichen aus der Gruppe R1 und die Kombination C2 mit Zeichen aus den Gruppen L3 und R1A. Diese Kombinationen definieren also die folgenden Zeichenfolgen:

```
C1 = db, dd, dp, dt; tb, td, tp, tt
C2 = bb, bd; pb, bd
```

C1 verbindet Zeichen aus Hauptkategorien (L1, R1), wohingegen C2 Zeichen aus einer Hauptkaterie (L3) mit einem Zeichen einer Unterkategorie (R1A) verbindet (weshalb es in C2 weniger Kombinationen gibt.

[68] Wenn wir dies nicht wollen, müssen wir für normale und tiefgestellte Vokale separate Gruppen definieren, was durchaus möglich ist. Es ist aber zu beachten, dass die Zahl der Kombinationen mit jeder zusätzlichen Gruppen exponentiell anwächst (deshalb der Versuch, möglichst wenige Gruppen zu definieren).

[69] Dies entspricht im Wesentlichen einem JSON-Format, bei dem aus Gründen der Lesbarkeit die Anführungszeichen und die geschweifte Klammer {} weggelassen werden. Intern konvertiert VSTENO die Definitionen in ein gültiges JSON-Format und parst sie mit der Funktion json_decode().

Beachten Sie bitte, dass es möglich ist, Kombinationen zu definieren, die sich überschneiden. Zum Beispiel:

```
C1:[L1,R1],
C3:[L1,R1A]
```

Aufgelistet entspricht dies den Zeichenfolgen

```
C1 = db, dd, dp, dt; tb, td, tp, tt
C3 = db, dd; tb, td
```

Solche Definitionen sind nicht per se unzulässig, denn Zeichenkombinationen definieren an sich noch keine Abstände (die geschieht erst später bei den Abstandsregeln). Dennoch ist hier Vorsicht geboten: Überschneidungen dieser Art können dazu führen, dass gleiche Zeichenfolgen mehrfach definiert werden (und dann hinsichtlich des zu verwendenden Abstandes zu einem Widerspruch führen). Hier könnte eine andere Bezeichnung helfen zum Beispiel:

```
C1:[L1,R1],
C1A:[L1,R1A],
C1B:[L1,R1B]
```

Dadurch wäre klar, dass C1A und C1B Unterkombinationen von C1 sind (wobei sich die Unterkombinationen C1A und C1B nicht überschneiden und Untermengen von C1 sind).

Prototypen

Schliesslich werden "Regelprototypen" definiert, welche die Kombinationen (in Verbindung mit den Vokalen) spezifischen Abständen zuordnen:

```
#BeginSubSection(session)
    "spacer_rules_list" := "
        R1:[C1,V1,D1,?],
        R2:[C1,V2,D2,],
        R3:[C2,V1,D3,?],
    R4:[C2,V2,D4,]";
#EndSubSection(session)
```

Eine erste Regel (R1) beschreibt hier die Zeichenkombinationen C1 in Verbindung mit Vokalen aus der Gruppe V1. Diese Kombination erhält den Abstand D1. Eine zweite Regel (R2) beschreibt die gleiche Zeichenkombination C1 in Verbindung mit den Vokalen aus der Gruppe V2. Diese Kombination erhält den Abstand D2 etc. Beachten Sie: Da wir 2 Vokalgruppen definiert haben, ergeben sich für die 4 Kombinationen die doppelte Anzahl also 8 Regeln. Das Zeichen ? bedeutet, dass der Vokal optional ist. Dies ist bei der Vokalgruppe V1 der Fall, da der Abstand zwischen dd (kein Vokal) und dad (Vokal a) genau gleich gross sein soll.

RX-GEN

Sobald Daten (Zeichengruppen, Zeichenkombinationen, Regelprototypen) wie oben beschrieben definiert haben, können wir daraus automatisch REGEX-Regeln generieren. Beachten Sie zuvor aber folgenden wichtigen Hinweise: Wenn Sie Definitionen im Header-Teil des Modells ändern, dann müssen Sie die nach dem Speichern (welche die Änderungen in die Datenbank schreibt) zusätzlich die SESSION-Variablen aktualisieren, indem Sie unterhalb des Formulars auf "aktualisieren" klicken. Tun Sie dies nicht, arbeitet RX-GEN mit den bestehenden (alten) Werten und sie erhalten falsche Resultate!

Klicken Sie nach dem Aktualisieren der SESSION-Variablen links unter "Tools" auf RX-GEN. Dieser "REGEX-Generator" liest nun die Zeichendefinitionen aus dem aktuell gewählten Modell (deutsch, spanisch oder custom) sowie aus den SESSION-Variablen, generiert daraus die Regeln und zeigt diese zusammen mit der Anzahl Zeichenkombinationen im Browser an. Als Beispiel[70]:

```
"(\[(?:in|sein|gegen|da|schaft|auf|aus|bei|bei+ar|durch|
solch|selb|wo|schm|ck|lt|rr|ss|sobre|tt|ch|zw|d|g|h|j|k|
s|s=|ser|t|v|w|x|y|z|&a|&u|&o|0n-|0d-|0u-|d@r|t@r|g@r|k@r|
ch@r|h@r|g@l3|v@l|w@l|t@l3|tt@l3|ck@l3|d@l3|k@l3|z@l3|
ch@l3|v@r6|w@r6|z@r|da@r|rr@l|ck@l|tt@r|ck@r|lt@r6)\])
(\[(?:a|o|u)\])?(\[(?:in|sein|gegen|hab|da|sind|haft|schaft|
auf|aus|-aus|bei|bei+ar|fort|schm|schw|mpf|sch|ll|mm|ss|
sobre|ch|pf|mp|zw|y|z|&e|&a|&u|&o|ch@r|sch@r|mm@l|pf@l|
z@l3|sch@l3|ch@l3|pf@r6|z@r|da@r)\])"
=> "$1[#d1]$2$3"; // R1|C1: L1#V1:D1#R1 (11256)
```

Wie man unschwer erkennt: Diese REGEX-Regeln werden schnell kompliziert und sind für einen Menschen kaum bzw. nicht mehr zu überblicken (zumal das gezeigte Beispiel nur eine von vielen Regeln ist, die generiert werden). Im konkreten Fall handelt es sich um die Regel R1, welche die Kombination L1+V1+R1 mit dem Abstand D1 definiert (es wird - um der Nachvollziehbarkeit willen - am Schluss der Regel immer angegeben, um welche Kombination es sich handelt). Diese Regel erstreckt sich auf 11256 Kombinationen.

Die von RX-GEN generierten und im Browser angezeigten Regeln können wir nun auswählen, kopieren und in unsere eigenen Regeln einfügen, zum Beispiel am Schluss in einer eigenen SubSection "spacer" (wie in den Beispielmodellen für Deutsch und Spanisch geschehen). Danach generiert die obige Regel dann unter anderem folgende Kombinationen:

```
[da@r][#d1][l]      Beispiel: darlegen
[t@r][#d1][vr]      Beispiel: trinken
```

Um den Abstand nun einfügen zu lassen haben wir nun zwei Möglichkeiten: (1) entweder wir definieren das Zeichen "#d1" als leeres Zeichen mit einem Breitewert

[70] Bitte beachten Sie, dass die Regel im Original keine Zeilenumbrüche enthält; diese wurden hier nur layouttechnisch eingefügt. Die gezeigte Regel entspricht nicht den obigen Beispielen sondern ist einem realen Modell (Spanisch) entnommen.

(Offset 1 im Zeichenheader), der den konkreten Abstand (in Pixeln) definiert, oder (2) wir definieren mit der gleichen Methode (Zeichenbreite im Offset 1) andere Zeichen - z.B. "#1", "#2", "#3" bzw. "#1-", "#2-", "3#-" etc. - mit fixen Abständen (hier also 1, 2, 3 bzw. -1, -2, -3 Pixel) und verwenden die Abstände "#d1", "#d2", "#d3" etc. nur als Platzhalter, die wir erst ganz am Schluss diesen fixen Abständen zuordnen. Die zweite Vorgehensweise hat einen entscheidenden Vorteil: Hier können nämlich die Abstände auch im Nachhinein noch angepasst werden, ohne die REGEX-Regeln noch einmal neu zu generieren. Dies kann zum Beispiel über REGEX-Regeln der folgenden Art geschehen:

```
"\[#d1\]" => "[#5]";
```

Hier wird dem Platzhalter "#d1" (der in unserer Gruppierung für D1 steht) also definitiv der Abstand 5 zugewiesen, da das Zeichen "#5" im Header als "leeres Zeichen mit Breite 5" definiert wurde.

Konkrete Beispiele finden sich im File SPSSBAS.TXT (Verzeichnis ling/). Hier wird das 2. Vorgehen angewandt, also Definition von Abstandszeichen "#1", "#2", "#3" etc. und Zuweisung derselben in der SubSection(spacer) am Ende (im Anschluss an die automatisch generierten Abstandsregeln).

Automatisieren

Die Generierung der Abstandsregeln lässt sich auch automatsieren. Dies ist besonders dann praktisch, wenn Sie an präzisen Zeichenabständen herumfeilen und die Abstandsregeln mehrfach abändern und testen wollen. VSTENO bietet hier im Formular Maxi unter Engine => Spacer die Option "automatisch". Wenn Sie diese anklicken generiert VSTENO bei jeder Berechnung neue Spacer-Regeln und fügt diese automatisch ins aktuelle Modell ein.

Damit dies funktioniert, müssen jedoch einige Regeln beachtet werden. Zum ersten muss der Regel-Teil des Modells in diesem Fall zwingend einen spacer-Teil enthalten, zum Beispiel in der folgenden Art:

```
#BeginSubSection(spacer)
    // no rules
#EndSubSection(spacer)
```

Dadurch weiss VSTENO, wo - d.h. zu welchem Zeitpunkt der Berechnung - er die Spacer-Regeln einfügen soll. Die Spacer-Regeln stellen also eine eigene SubSection dar, die durch VSTENO vollständig ersetzt wird. Fall die Spacer-Section zusätzliche Argumente enthält - z.B. eine Zuweisung oder eine Verzweigung der Art #EndSubSection(spacer,=:prt) - so bleibt diese erhalten.

Wie bereits im vorhergehenden Abschnitt erwähnt, müssen Sie bei Änderungen in den Spacer-Definition zwingend die SESSION-Variablen aktualisieren, indem Sie unten im Formular auf "aktualisieren" klicken. Sonst verwendet VSTENO die bestehenden (alten) Werte und Sie erhalten falsche Resultate (was die ohnehin schwierige Fehlersuche zusätzlich erschwert).

Die Generierung von Spacer-Regeln on-the-fly ist eine gute Möglichkeit, um präzise Formeln zu entwickeln. Allerdings wirkt sich dies schlecht auf die Performance des Programmes aus. Deshalb sollten Sie am Schluss, wenn Ihre Regeln also fertig sind, die Regeln wie bereits beschrieben mit RX-GEN generieren und fix innerhalb Ihrer Regeln einfügen, also:

```
#BeginSubSection(spacer)
    // insert
    // your
    // spacer
    // rules
    // here
#EndSubSection(spacer)
```

Beachten Sie bitte, dass sie hier KEINE handgeschriebenen, sondern NUR die automatisiert generierten Regex-Regeln einfügen dürfen, wenn Sie handgeschriebene Regeln benötigen, die zum Spacer gehören, und vor oder nach den automatisierten Regeln ausgeführt werden soll, dann erstellen Sie hierfür separate SubSections - zum Beispiel #BeginSubSection(prespacer) - #EndSubSection(prespacer) und #BeginSubSection(postspacer) - #EndSubSection(postspacer) - und fügen Sie die Regeln dort ein. Dadurch ist sichergestellt, dass VSTENO nur automatisch generierte Regeln ersetzt, wenn die Option "automatisch" gewählt ist.

Grundsätzlich wird empfohlen, immer ein Spacer-Section vorzusehen und diese mit gültigen statischen Regeln zu füllen.

Linguistische Analyse

Die linguistische Analyse geht vom Schriftbild des Textes aus und bedient sich grundsätzlich dreier externer Programme mit unterschiedlichen Aufgabenbereichen:

1. phpSyllable: Silbentrennung
2. hunspell: Erkennung zusammengesetzter Wörter und Affixe (Vor- und Nachsilben)
3. eSpeak: Phonetische Transkription

Die Verwendung der linguistischen kann zunächst einmal über das Eingabe-Formular "Maxi" unter "Sprache" eingestellt bzw. definiert werden:

- Option Analyse: keine = die Analyse bleibt ausgeschaltet; Auswahl = die externen Programme (1-3) können einzeln an- und abgewählt werden.
- Programmoptionen
  - Silbentrennung: Es kann die Sprache angegeben werden (de: Deutsch, es: Spanisch, fr: Französisch, en: English).

- Wörter/Affixe: Es kann ebenfalls die Sprache angegeben werden[71]. Ausserdem kann gewählt werden, ob Affixe grundsätzlich generiert werden sollen (Option Affixe) oder ob auf die Markierung einzelner Affixtypen (Präfixe, Suffixe) oder von Wortgrenzen verzichtet werden soll (in diesem Fall "keine" anwählen und auszuschliessende Markierungen selektieren).
- Phonetik: Es kann wiederum die Sprache angegeben werden sowie das Alphabet, das für die Transkription verwendet werden soll (eSpeak = Kirschenbaum Alphabet[72] oder IPA = Internationales Phonetisches Alphabet).

Obwohl es sich um drei externe und unabhängige Programme handelt, bestehen zwischen ihnen im Kontekt von VSTENO Abhängikeiten, die dazu führen, dass nicht alle Kombinationen möglich sind bzw. Sinn machen. Empfohlen werden folgende Kombinationen:

1. Keine linguistische Analyse: Das Spansiche Modell SPSSBAS zum Beispiel verwendet keine linguistische Analyse, da die spanische Orthografie sehr regelmässig und Spanisch - im Unterschied zum Deutschen - nur wenige zusammengesetzte Wörter enthält. Die Regeln können dadurch direkt (und effizient) beim Schriftbild ansetzen (ohne sich zum Beispiel um Silbengrenzen zu kümmern). Ein Modell, das keine linguistische Analyse benötigt, ist auch sehr schnell.

2. Nur Silbentrennung: In diesem Fall ruft VSTENO nur phpSyllable auf, um eine Silbentrennung durchzuführen. Silben werden als - in den Text eingefügt und die Regeln müssen dies berücksichtigen. Auch hier kann mit dem Schriftbild gearbeitet werden und Übertragung ist recht schnell.

3. Silbentrennung + Wort/Affixanalyse: In diesem Fall ruft VSTENO zuerst phpSyllable auf, um Silben zu generieren, und verwendet danach hunspell (und einen speziellen Algorithmus[73]) um zusammengesetzte Wörter und Affixe zu erkennen. Diese Analyse ist relativ langsam und es gibt grundsätzlich zwei Varianten

   (a) Vollanalyse: Präfixe (+), Suffixe (#), Silben (-) und Wortgrenzen (|) werden mit den enstprechenden Zeichen markiert und die Regeln müssen diese berücksichtigen.

   (b) Teilanalyse: Präfixe (+), Suffixe (#) und Silben (-) zum Beispiel werden abgewählt: in diesem Fall müssen sich die Regeln nur noch um die verbleibenden Markierungen (Wortgrenzen = |) kümmern.

[71] Dies funktioniert im Prinzip gleich wie bei der Silbentrennung. Allerdings bietet hunspell nebst der Wahl der Hauptsprache (zum Beispiel "de" für Deutsch) zusätzlich regionale Varianten an (zum Beispiel "de_CH" für in der Schweiz verwendete Schreibweisen; "de_CH" ist die Standardeinstellung für das Modell DESSBAS).

[72] Siehe hier: https://en.wikipedia.org/wiki/Kirshenbaum. Das Kirschenbaum-Alphabet hat den Vorteil, dass es ASCII-kompatibel ist, also mit jeder Tastatur zuverlässig eingegeben und auf jeder Textkonsole angezeigt werden kann. Das französische Modell FRSSBAS verwendet das Kirschenbaum-Alphabet.

[73] Siehe hier: Gel speiende Spiegel und andere (linguistische) Probleme

4. Nur Phonetik: Der Text wird vom Schriftbild ins Lautbild übertragen. Die nachfolgenden Regeln haben deshalb keine Möglichkeit mehr, sich am Schriftbild zu orientieren[74]. Diese Art der Übertragung ist mittelschnell.

Wichtig ist somit anzumerken, dass insbesondere folgende Kombinationen NICHT möglich sind: (A) Wort/Affixanalyse ohne Silbentrennung (da die Wort/Affixanalyse auf der Silbentrennung basiert) und (B) Phonetische Analyse in Verbindung mit den anderen beiden Analysen (es kann also nur die phonetische Analyse allein verwendet werden).

Vorselektion

Wenn Sie selber ein Modell (= stenografisches System) mit VSTENO umsetzen, macht es natürlich Sinn, die für das Modell notwendigen Optionen direkt vorzuselektieren, dies kann geschehen, indem Sie im HEADER-Teil des Modells folgende Variablen direkt setzen. Hier das Beispiel für das deutsche System DESSBAS:

```
#BeginSubSection(session)
    // use linguistical analysis
    "analysis_type" := "selected";
    // hyphenation (phpSyllable)
    "hyphenate_yesno" := "yes";
    "language_hyphenator" := "de";
    // composed words + affixes (hunspell)
    "composed_words_yesno" := "yes";
    "language_hunspell" := "de_CH";
    // generate affixes in composed words
    "affixes_yesno" := "yes";
    // set variables for filter option (not selected)
    "filter_out_prefixes_yesno" := "yes";
    "filter_out_suffixes_yesno" := "yes";
    "filter_out_words_yesno" := "yes";
    // disable phonetic analysis
    "phonetics_yesno" := "no";
    "language_espeak" := "de";
    "phonetic_alphabet" := "espeak";
#EndSubSection(session)
```

Hier werden also die weiter oben genannten Einstellungen direkt mit "yes" oder "no" selektiert bzw. die Sprache voreingestellt. Dies führt dazu, dass VSTENO diese Werte automatische lädt, wenn im Eingabeformular "Maxi" das Modell angewählt wird. Beachten Sie bitte, dass die Zuweisen mit dem Operator := erfolgt und dass sowohl Variablen wie Werte in doppelten Anführungszeichen stehen müssen.

[74] Wie wir später sehen werden, besteht die Möglichkeit in der Stage 0 gewisse Wörter von der Transkription auszunehmen, indem sie mit #wort markiert werden. Die Regeln des Modells basieren aber zum grössten Teil - und ausschliesslich - auf dem Lautbild der Wörter.

Im folgenden werden wir nun auf diese linguistischen Analysen genauer eingehen. Da die phonetische Analyse nur separat verwendet werden kann macht, es Sinn, die Betrauchtung auf zwei Kapitel zu verteilen. Ausschlaggebend ist dabei, dass am Schluss der Analyse für die Regeln zur Verfügung steht, nämlich entweder eine mehr oder weniger (A) Morphologische Analyse oder als (B) Phonetische Transkription des Textes.

Morphologische Analyse

Die von VSTENO verwendete morphologische Analyse verwendet die externen Programm phpSyllable (Silbenerkennung) und hunspell (Wort/Affixanalyse) und basiert mehr oder minder auf einem speziellen Algorithmus, der hier genauer erläutert wird. Als einleitende und erste grobe Übersicht ist wichtig zu wissen:

1. Dass die linguistische Analyse automatisch in Stage 1 vorgenommen wird.
2. Dass die linguistische Analyse fehlerhafte Resultate produzieren kann.
3. Wie die linguistische Analyse angepasst werden kann, damit möglichst wenige falsche Resulate generiert werden.

Der Reihe nach:

Zu Punkt 1: Es ist zu präzisieren, dass die linguistische Analyse nur dann vorgenommen wird, wenn kein Eintrag im Wörterbuch gefunden wird (sonst wird die LNG-, STD- oder PRT-Form aus dem Wörterbuch verwendet).

Zu Punkt 2: Zusammengesetzte Wörter sowie Silben- und Morphemgrenzen zu erkennen ist für einen Computer alles andere als einfach. Der Computer findet zuweilen absurde "Kombinationen" wie: Spiegel (Verb "spie" + Substantiv "Gel"), Museum (Substantiv "Muse" + Präposition "um"), anderen (Präposition "an" + Relativpronomen "deren") etc.

Zu Punkt 3: Es bestehen grundsätzlich zwei Möglichkeiten, falsche Resultate zu vermeiden: (1) indem man die linguistische Analyse mit zusätzlichen Daten zu Präfixen, Stämmen und Suffixen versorgt und (2) indem man falsche Resultate mit REGEX-Regeln korrigiert. Beides wird im folgenden kurz erläutert.

Parameter und Variablen

Das Verhalten der linguistischen Analyse durch Parameter gesteuert werden, die über die Eingabemaske "Maxi" verfügbar sind. Die Variablen beziehen sich auf das verwendete Wörterbuch (hunspell), den Silbentrenner (phpSyllable) sowie den Algorithmus, der Vor- und Nachsilben analysiert[75]. Die veränderbaren Parameter (wie Sprache und Auswahl gezielter Elemente der Analyse) wurden bereits in den beiden vorhergehenden Kapiteln erklärt[76].

[75] Funktion analyze_word_linguistically() in linguistics.php, welche die einzelnen Analyse-Schritte zusammnefasst und kombiniert.

[76] Nachzureichen wären hier nur noch die Optionen "Trennen" (separate) und "Leim" (glue): Diese wurden ursprünglich verwendet, um anzugeben, bei welcher Länge Wörter abgetrennt oder

Im Zusammenhang mit der morphologischen Analyse interessiert uns nun die Möglichkeit, Präfixe, Stämme und Suffixe zu definieren und markieren zu lassen. Hierzu muss im Header-Teil ein SubSection "session" definiert werden[77]:

```
#BeginSubSection(session)
  "prefixes_list" := "an, ver, ge";
  "stems_list" := "gan-?gen, nann-?t(?:e[rsnm]?)?";
  "suffixes_list" := "[kh]ei-?t(?:s|en)?,
  li-?ch(?:(?:e-?r)(?:e[srn]?)?)?";
#EndSubSection(session)
```

Session bedeutet hier nichts anderes, als dass wir Session-Variablen des Programmes verändern können (wie bei Inline-Option-Tags). Die Session-Variablen für Präfixe, Stämme und Suffixe heissen prefixes_list, stems_list und suffixes_list. Gesetzt werden die Werte durch den Operator :=[78] und die einzelnen Präfixe, Suffixe und Stämme stehen innerhalb von Anführungszeichen und mit Komma (falls es mehrere sind). Im obigen Beispiel werden also drei Präfixe definiert (an-, ver- und ge-), sowie die zusätzlichen (unregelmässigen) Stämme -gangen (z.B. in "gegangen") und -nannte in verschiedenen Varianten (genannte, genannter, genanntes etc.) und letztlich auch die Suffixe -keit/-heit und -lich in verschiedenen Varianten.

Wie leicht zu erkennen ist, kann auch hier der REGEX-Formalismus verwendet werden, allerdings gibt es eine absolut wichtige Einschränkung zu beachten: Es dürfen nur so genannte "non capturing groups" - Zeichen (?:) statt () - verwendet werden. Der Grund hierfür ist, dass VSTENO diese Formeln zur Abarbeitung in eine übergeordnete Formel ("Metaformel") einbettet: dies Metaformel erkennt die einzelnen Elemente (Vor-/Nachsilben, Stämme), kopiert (trennt) die entsprechenden Wortteile und fügt sie wieder zusammen (versehen mit den Markern für Wort-, Silben-, Morphemgrenze). Hierfür - für das Trennen/Kopieren/Einfügen - setzt die Metaformel ihrerseits capturing groups ein, um die einzelnen Teile richtig zuzuordnen. Werden nun in den Definitionen für Präfixe, Stämme und Suffixe zusätzliche capturing groups verwendet, so werden die Variablen falsch zugewiesen und das Chaos ist vorprogrammiert!

Was bewirken nun die obigen Zeilen? Wir erläutern dies wiederum anhand einiger Beispiele:

- Präfixe: Die linguistische Analyse von VSTENO wird als erstes eine Silbentrennung (phonetische Analyse) und dann eine Worterkennung (lexikalische Analyse) vornehmen. Dadurch wird das Wort "angeben" zum Beispiel zuerst in 3 Silben "an-ge-ben" aufgetrennt und danach als zwei separate Wörter "an|geben" markiert (hier kommt die lexikalische Analyse zum Zug, die sowohl "an" als auch "geben" als existierende Wörter identifiziert). Dank der prefix_list, weiss die linguistische Analyse nun zusätzlich, dass "an" am Wortanfang als

nicht abgetrennt (d.h. als eigenständige Wörter betrachtet) werden sollten. Diese vorgehen hat sich aber nicht bewährt und die Optionen werden voraussichtlich in einer späteren Version wieder entfernt.

[77] Im Original enthält der Text keine Zeilenumbrüche, diese wurden nur layouttechnisch eingefügt.

[78] Pascal lässt grüssen ... :)

Präfix zu behandeln ist. Also wird in einem dritten Schritt "an|ge-ben" zu "an+ge-ben" umgewandelt. Bitte beachten Sie, dass Präfixe nur dann als Präfixe behandelt werden, wenn sie zuvor als separate Wörter erkannt wurden: In "angeln" zum Beispiel wird kein Präfix erkannt (weil *geln kein eigentständiger Wortteil ist[79]). Etwas anders funktioniert das Wort "gegeben": Während bei "angeben" die Vorsilbe "an" als eigenes Wort erkannt wird (weil "an" als Präposition im Wörterbuch von hunspell exstiert), ist dies bei "ge" nicht der Fall. Hier weiss VSTENO als nur dank der prefix_list, dass ge als Vorsilbe betrachtet werden kann.

- Stämme: Bei "angeben" und "gegeben" funktioniert die linguistische Analyse, wie wir gesehen haben, sehr gut (weil "geben" im Wörterbuch existiert). Betrachten wir das Wort "gegangen" hingegen, so wird VSTENO zunächst keine Vorsilbe finden (da *gangen kein eigentständiges Wort ist). Auch hier können wir der linguistischen Analyse auf die Sprünge helfen, indem wir "gangen" als eigenständigen Stamm definieren. Dadurch schlagen wir gleich zwei (oder zuweilen noch mehr) Fliegen auf eine Klappe: Nicht nur "ge+gangen" wird nun erkannt, sondern auch "ver+gan-gen" (weil ver- in der prefix_list steht).

- Suffixe: Funktionieren nach dem gleichen Prinzip wie Präfixe, mit dem einzigen Unterschied, dass sie durch # abgetrennt werden: "som-mer#lich". Im Unterschied zu Präfixen können sich Suffixe verändern (z.B. durch Konjugation, Deklination). Hier hilft uns REGEX, um die verschiedenen Fälle abzudecken: "li-?ch(?:(?:e-?r)(?:e[srn]?)?)?" erkennt "som-mer#lich", "sommer#lich-e", "sommer#lich-es", "som-mer#lich-er" etc. Bitte beachten Sie auch hier die Verwendung von non capturing groups (?:).

Sie ahnen es vielleicht anhand der gezeigten Beispiele: Die Sache kann ziemlich schnell kompliziert werden, sodass man hier wirklich sehr gute Regeln austüfteln muss, um optimale Resultate zu erhalten. Dennoch ist die linguistische Analyse und das "Kalibrieren" derselben mit zusätzlichen Variablen ein sehr potentes Mittel, um Wortteile mit guter Trefferquote zu erkennen - und dadurch das Formulieren der nachfolgenden Stenoregeln zu vereinfachen!

Aber so gut man die Regeln auch wählt, es werden immer falsche Resultate generiert werden. Diese können nun aber auf zwei Arten korrigiert werden: (1) mit der Blockliste/Filterliste oder (2) mit einem sogenannten Postprocessing (Nachbearbeitung mithilfe von REGEX-Regeln).

Blockliste / Filterliste

Blocklist und Filterlist sind Session-Variablen die genau gleich gesetzt werden können wie Präfixe, Suffixe und Stämme, nämlich:

```
#BeginSubSection(session)
  "block_list" := "des, der, den, dem";
```

[79] In "Angel" hingegen wird ein falsches Präfix erkannt, weil "an" und "Gel" eigentständige Wortteile sind. Dies ist einer der Fälle, die nachträglich korrigiert werden müssen.

```
  "filter_list" := "de[snrm]";
#EndSubSection(session)
```

Hier wird also der Blockliste eine Liste mit 4 Wörtern (des, der, den, dem) zugeordnet, die "blockiert" werden sollen. Als zweites Beispiel wird der Filterliste 1 REGEX-Ausdruck (de[snrm]) zugeordnet (der im Prinzip die gleichen 4 Wörter - des, den, der, dem - abdeckt - es wird anstelle der Aufzählung einfach ein REGEX-Ausdruck verwendet).

Was aber bewirken nun diese Angaben? Ein Eintrag in der Blockliste bedeutet grundsätzlich: VSTENO wird diese Buchstabenfolge nicht mehr als eigenes Wort betrachten. Nehmen wir als Beispiel das Wort "lachendes". VSTENO wird dieses in Silben trennen (la-chen-des) und schliesslich versuchen, "Teilwörter" (also zusammengesetzte Wörter) zu finden. Dazu generiert es zunächst eine Liste sämtlicher möglicher Silbenpermutationen, in diesem Fall also Kombinationen mit ein, zwei oder drei Silben (la, chen, des; lachen, chendes; lachendes). Danach verwendet die linguistische Analyse hunspell und die Präfix-, Stämme-, Suffixlisten, um zu entscheiden, welche Kombinationen als eigene Wortteil existieren bzw. möglich sind. Alle anderen Möglichkeiten werden "gestrichen". In unserem Fall ergäbe das also: -, -, des; lachen, -; lachendes. Aufgrund dieser Analyse kann VSTENO schliesslich annehmen, dass das Wort "lachendes" aus zwei Teilwörtern (lachen + des) besteht und markiert dieses mit einer Wortgrenze (lachen|des).

Fügen wir nun "des" zur Blockliste hinzu, wird auch "des" aus der Liste gestrichen und es bleibt somit nur noch: -, -, -; lachen, chendes; lachendes. VSTENO findet in diesem Fall also keine kleinere Unterteilung des Wortes und wird deshalb "lachendes" als ein einziges, unaufteilbares Wort betrachten - was in diesem Fall richtig ist.

Ähnlich verfährt VSTENO dann auch mit dem Wort "deswegen": Die Silbentrennung ergibt: des-we-gen. Die möglichen Permutationen somit: des, we, gen; deswe, wegen; deswegen. Streichung mit hunspell: des, -, -; -, wegen; deswegen. Zusätzliche Streichung mit der Blockliste: -, -, -; -, wegen; deswegen. Auch hier kommt VSTENO zum Schluss, dass "deswegen" keine Teilwörter mehr enthält - und wird somit auch keine Präfix "des" markieren. In diesem Fall ist das schade, denn das Wort enthält tatsächlich die Vorsilbe des- mit entsprechender Kürzung im System Stolze-Schrey.

Kurzum: Zwar konnte wir im Fall von "lachendes" ein richtiges Resultat produzieren, dafür erkennt VSTENO nun die Vorsilbe in "deswegen" nicht mehr. Aus diesem Grund gibt es die Filterliste. Wenn wir im Falle von "des" beide Fälle abdecken wollen, dann empfiehlt sich folgendes Vorgehen:

1. "des" wird nicht zur Blockliste hinzugefügt => dadurch wird es als eigenständiger Wortteil erkannt

2. "des" wird zur Präfix-Liste hinzugefügt => dadurch wird der selbständige Wortteil als Präfix markiert

3. Wir fügen nun den REGEX-Ausdruck "|de[rsmn]" zur Filterliste hinzu => dadurch wird das falsche Resultat (false positive) "la-chen|des" korrigiert (und wieder zu "la-chen-des")

Bitte beachten, dass in der Filterliste nur eine eingeschränkte REGEX-Notation verwendet werden darf. Grundsätzlich dürfen das Zeichen |, eckige Klammern [] und Quantifier wie *+? verwendet werden, jedoch - wie bereits in der Präfix-Liste keine "markierenden" runden Klammern und keine Variablen (z.B. $1).

Postprocessing

Ich gebe zu, dass ich Ihnen oben eine etwas heile Welt vorgegaukelt habe: das Beispiel "angeben" würde in Wahrheit nämlich nicht zu "an+ge-ben" aufgelöst (wie oben angegeben), sondern zu "an+ge+ben". Warum? Ganz einfach: Weil Ben ein Eigenname ist, der im Wörterbuch vorkommt ...[80] VSTENO wird also denken[81]: Wenn "Ben" existiert und "ge-" eine Vorsilbe sein kann, dann muss (weil auch an- als selbständiges Wort existiert) auch ge- ein Präfix sein![82]

Kurzum: Die linguistische Analyse wird uns mit dem falschen Resultat "an+ge+ben" beglücken. Hier setzt nun das Postprocessing (Nachbearbeitung) an, die als Sub-Section mit dem Namen analyzer ebenfalls im Header stehen muss:

```
#BeginSubSection(analyzer)
   "(\||\+)(ben)$" => "-$2";
#EndSubSection(analyzer)
```

Diese Regel definiert nun, dass "ben" am Wortende nie als eigenständiges Wort (Eigenname) betrachtet wird, wenn voraus eine Wort- oder eine Morphemgrenze steht. Mit anderen Worten: "an+ge+ben" wird umgeschrieben zu "an+ge-ben" (Silbengrenze). Diese Regel hat noch weitere ungeahnte Auswirkungen: "sie|ben" zum Beispiel (ja, auch hier hat VSTENO geflissentlich[83] ein Pronomen "sie" und den Eigennamen "Ben" erkannt. Auch dies wird nun "zurückgeschrieben" zu "sie-ben".

Postprocessing Regeln sind also REGEX-Regeln, die im Header stehen und - ähnlich wie in Stage 2 und 4 - auf das ganze Wort (inklusive Satzzeichen, Morphem- und Silbengrenzen) angewendet werden!

Phonetische Analyse

In einigen Sprachen wie zum Beispiel Französisch, wo Laut und Schrift stark voneinander abweichen, ist es unumgänglich auf eine phonetische Transkription zurückgreifen zu können. VSTENO ermöglicht dies durch die Verwendung des Programmes eSpeak[84], das ebenfalls unter der freien Sofwarelizenz GPL herausgegeben wurde. Das Programm eSpeak ist eigentlich ein Sprachsynthese-Werkzeug, kann als solches

[80] Man sieht: Computer sind hier sehr kreativ - oder eben einfach nur dumm ... ;-)

[81] Bzw. "rechnen", wie gesagt, denn es ist ein Computer ... ;-)

[82] Man wäre hier vielleicht versucht, das Problem dadurch zu lösen, indem man definiert, dass Präfixe strikte am Wortanfang stehen müssen, aber das wird der Problemstellung nicht gerecht: "angegeben" zum Beispiel - hier werden die Vorsilben an- und ge- tatsächlich kombiniert (und es können auch noch mehr Vorsilben kombiniert werden: "unangefochten" zum Beispiel hat deren drei ...). Sprache (und hier insbesondere Präfixe) ist (sind) also tatsächlich rekursiv.

[83] ... aber ohne zu wissen, was es tut ... ;-)

[84] http://espeak.sourceforge.net/

aber - im Quiet-Modus - auch phonetische Transkriptionen vornehmen. Die phonetische Analyse via eSpeak kann in VSTENO durch folgende Session-Variablen im Header-Teil des Modells vorselektiert werden:

```
#BeginSubSection(session)
    // use linguistical analysis
    "analysis_type" := "selected";
    // hyphenation (phpSyllable)
    "hyphenate_yesno" := "no";
    "language_hyphenator" := "fr";
    // composed words + affixes (hunspell)
    "composed_words_yesno" := "no";
    "language_hunspell" := "fr";
    // generate affixes in composed words
    "affixes_yesno" := "no";
    // set variables for filter option (not selected)
    "filter_out_prefixes_yesno" := "no";
    "filter_out_suffixes_yesno" := "no";
    "filter_out_words_yesno" := "no";
    // disable phonetic analysis
    "phonetics_yesno" := "yes";
    "language_espeak" := "fr";
    "phonetic_alphabet" := "espeak";
#EndSubSection(session)
```

Beachten Sie, dass es nicht möglich ist, eSpeak mit dem Silbentrenner (phpSyllable) und der Wortanalyse (hunspell) zu kombinieren. Es empfiehlt sich, bei der Vorselektion der linguistischen Analyse immer alle Variablen zu setzen (also zum Beispiel auch "fr" für die Variablen "language_hyphenator" und "language_hunspell" etc. auch wenn Sie nicht verwendet werden[85]).

Wir können uns nun mit den obigen Einstellungen anschauen, wie eine erste französische Transkription aussieht. Wir wählen hierfür das Wort "renseignement". Die Wahl von LNG als Ausgabeform ergibt nun:

```
rA~sEnj@mA~
```

Dies entspricht der Ausgabe "espeak" die wir mit der Variable "phonetic_alphabet" vorselektiert haben. Verwendet wird hier grundsätzlcih das so genannte Kirshenbaum Alphabet[86]. Der Vorteil des Kirshenbaum-Alphabets besteht darin, dass nur Zeichen verwendet werden, die auf einer normalen Tastatur einfach eingegeben und auch in einem normalen ASCII-Text-Editor gut dargestellt werden können. Der Nachteil besteht darin, dass für ungewöhnlich Laute manchmal mehrere Zeichen mit Gross-

[85] Die Parameter können ja auch vom/von der Benuzter/in im Eingabe-Formular Maxi verändert werden; das Setzen aller Variablen steht sicher, dass dem/der Benuzter/in korrekte Parameter zur Auswahl angeboten werden.

[86] https://en.wikipedia.org/wiki/Kirshenbaum

und Kleinschreibung verwendet werden. So z.B. die Kombination "A~" für das nasale A, "E" für ein offenes E, "@" für stummes E (e muet) etc.[87]

Im Prinzip kann man man nun mit diesen Zeichen genau gleich Regeln formulieren, wie das mit dem Schriftbild der Fall war, also zum Beispiel:

```
“mA~$” => “{MENT}”;
"^kO~tr" => "{CONTRE}";
"^Etr$" => "{ETRE}";
```

Für die Kürzungen -ment, contre-, être. Allerdings ist Französisch eine vertrackte Sprache: da es viele Homonyme (gleichlautende Wörter) gibt, wird oft das Schriftbild dazu verwendet, um sie zu unterscheiden:

```
son/sont <=/=> le son
sans <=/=> s’en/cent/c’en/sang
```

Hier sollen für "son/sont" und "sans" die entsprechenden Kürzungen verwendet werden, "le son" und "c'en/cent/c'en/sang" sollen jedoch ausgeschrieben werden.

Dies kann dadurch gelöst werden, dass man solche Wörter zunächst in der Stage 0 - wo ja noch das Schriftbild des Wortes zur Verfügung steht - vormarkiert:

```
"(?<=^| )([Ll]e|[Uu]n) son(?= |$)" => "$1_son"; // R1
"(?<=^| )[Ss]ont?(?= |$)" => " #son ";   // R2
"([Ll]e|[Uu]n)_son" => "$1 son";                 // R3
"(?<=^| )[Ss]ans(?= |$)" => "#sans";             // R4
```

Betrachten wir zuerst den Fall "son/sont" vs "le son": Die Abgrenzung "son" vs "sont" ist sehr einfach (hier hilft das unhörbare, aber geschriebene t). Diese Unterscheidung "son" (Personalpronomen) vs "son" (Substantiv) hingegen ist sehr knifflig. Wir können hier versuchen, syntaktische Elemente herbeizuziehen, z.B. ob "son" ein Artikel, also "le" oder "un" vorausgeht. Dies tut nun die erste Regel R1 (indem es im Gesamttext - Stage 0 - nach solchen Kombinationen sucht) und schreibt diese Kombinationen dann als "le_son" oder "un_son". Nun kann die zweite Regel ganz einfach alle verbleibenden, alleinstehenden (das heisst von einem Space-Zeichen umgebenen "son") zu #son umschreibt. In einem dritten Schritt (R3) schreiben wir "le_son" und "un_son" wieder zu "le son" und "un son" (getrennt geschrieben zurück).

Die Markierung von #son hat nun zur Folge, dass eSpeak dieses Wort nicht phonetische umschreiben wird. Das Wort #son gelangt also unverändert in die Stage 1 (Nachschlagen im Wörterbuch), Stage 2 (einzelne, ganze Wörter; mehrere Falls zusammengesetzt) und Stage 3 (einzelne Teilwörter). Dort können wir sie nun mit dem Shortener zum Beispiel definitiv als Kürzung umschreiben:

[87] Alternativ kann das Internationale Phonetische Alphabet (IPA) verwendet werden. Es ist aber unklar, inwieweit es realistisch ist, basierend auf diesem Alphabet ein funktionierendes stenografisches Modell mit einem simplen Texteditor zu erstellen (und in der Datenbank, die punkto Sonderzeichen ebenfalls etwas "reizbar" ist) abzuspeichern. An dieser Stelle soll deshalb ganz förmlich von der Verwendung des IPA abgeraten werden ... (Aber wer es versuchen möchte, darf natürlich ... ;-)

```
"#son" => "{SON}";
```

Das nicht markierte Wort "le son" hingegen wird phonetisch zu "l@ sO~" transkribiert und kann danach anhand der System-Regeln zu einem Stenogramm umgeschrieben werden.

Etwas einfach funktioniert das Wort "sans": Da dieses unveränderlich ist, dann es einfach als alleinstehendes (= zwischen Spaces stehendes) Wort erkannt und mit # als "#sans" markiert werden. Alle anderen, vom Schriftbild her verschiedenen, aber lautlich gleichen Varianten wie "c'en/cent/s'en/sans" werden von eSpeak phonetisch zu "sA~" transkribiert und gemäss der Standardregeln in ein Stenogramm übertragen.

Beachten Sie bitte, dass die obige Regeln nicht der Weisheit letzter Schluss sind. Sämtliche Wörter die in Anführungszeichen stehen oder die von einem Satzzeichen (,.:;!?) funktionieren zum Beispiel nicht[88]. Auch erkennt die Regel R1 zwar zuverlässig "le son" und "un son" nicht aber "un dernier son". Kurzum: Regeln zu schreiben, die hier allen möglichen Fällen gerecht werden, ist eine sehr knifflige Angelegenheit!

Bitte beachten Sie auch, dass es bis zu einem gewissen Grad auch möglich ist, markierte Wörter mit Teilen zu kombinieren, die phonetisch behandelt werden sollen. Beispielsweise gibt es im Französischen viele Wörter mit e muet, die mittels Apostroph mit einem vokalisch anlautenden Folgewort verbunden sind: m'étais, l'avait, n'avions etc. Die Regel in Stage 0

```
"(?<=^| )[Aa]v(ai[st]|ions|iez|aient|oir)(?= |$)" => "#av$1";
```
[89]

schreibt zum Beispiel "n'avions" in "n#avions" um. Im Shortener-Teil können wir dann die Regel

```
"#avions" => "{AV}j[o~]";
```

einfügen. Das Stenogramm wir schliesslich korrekt generiert:

```
(1) n stammt aus der Regel in Stage 0 und entspricht phonetisch n
(2) {AV} wurde in Stage 0 als "#avions" vormarkiert und im Shortener (Stage 3) ersetzt
(3) lautliches "j[o~]" wird vom Shortener als "phonetischer Rest zur Kürzung {AV}" umge
```

Dies nur einige Beispiele, die zeigen sollen, dass durch geschickte Kombinierung der Stages in VSTENO sehr vieles möglich ist.

[88] Dies kann man z.B. wiederum dadurch lösen, dass man zuerst künstlich in Stage 0 Abstände einfügt, die man danach - weiterhin in Stage 0 - wieder herausfiltert, also: "(\(|\[|\"|'|\?|!)([a-zA-Zàâêéèîôûù])" => "$1#>#$2"; und "([a-zA-Zàâêéèîôûù])(\)|\]|\"|'|!|\?)" => "$1#<#$2"; etc.

[89] Damit diese Regel funktioniert, müssen bei Apostrophes zunächst künstliche Leerzeichen eingefügt werden, wie später wieder herausgefiltert werden (wie weiter oben beschrieben).

Wörterbuch

In Stage 1 schickt VSTENO das zu berechnende Wort zuerst zum Wörterbuch. Dort wird es entweder gefunden oder nicht. Wenn es gefunden wird, enthält das Wörterbuch drei Einträge namens LNG (linguistical form), STD (standard form) und PRT (print form). Der Unterschied zwischen STD und PRT ist grundsätzlich, dass STD eine Standard-Version von Stenogrammen darstellt. PRT wiederum entspricht quasi 1:1 dem Stenogramm, wie es am Ende generiert werden soll[90].

Es können nun drei Fälle auftreten:

1. Das Wörterbuch enthält nur LNG
2. Das Wörterbuch enthält sowohl LNG als auch STD
3. Das Wörterbuch enthält sowohl LNG als auch STD als auch PRT[91]

Ist das Wort in der PRT Form enthalten, werden keine weiteren Regeln mehr abgearbeitet, sondern direkt ein Stenogramm generiert. Ist das Wort in der STD Form enthalten, wird die STD-Form vom Parser in die PRT überführt, indem er einen Teil der Regeln anwendet. Ist nur die LNG-Form enthalten, wird zuerst die STD- und danach die PRT-Form generiert. Damit dies funktioniert, muss im Parser eine Funktion mit folgendem Format definiert sein:

```
#BeginSubSection(bundler)
     // Regeln
#EndSubSection(bundler,=:std)
```

Das Symbol "=:" bedeutet hier "weise zu". Die Zuweisung erfolgt hier der Variable std, welche der STD-Form entspricht. Jede Regelsammlung, die ein Stenografie-System definiert, sollte also eine Funktion enthalten, wo im #EndSubSection-Teil der Variablen STD ein Wert zugewiesen wird (damit diese für das Wörterbuch verwendet werden kann).

Das Gleiche gilt für die PRT-Form, die ganz am Ende der Regelsammlung zugewiesen werden sollte:

```
#BeginSubSection(spacer)
    // Regeln
#EndSubSection(spacer,=:prt)
```

Sowohl STD- als auch PRT-Form sollten erst in Stage 4 (und keines falls in Stage 3) zugewiesen werden (da in Stage 3 bei zusammengesetzten Wörtern nur Teilwörter bearbeitet werden und die Formen deshalb unter Umständen nicht vollständig sind).

Die Zuweisung der LNG-Form erfolgt in VSTENO automatisch: Sie ist jene Form, die direkt der linguistischen Analyse + Postprocessing entstammt.

[90] Weitere Hinweise zur STD und PRT Form finden sich hier: https://www.vsteno.ch/docs/mitmachen_bei_vsteno.pdf

[91] Die Formen LNG, STD, PRT entsprechen der chronologischen Abfolge der Berechnung, deshalb existiert im Prinzip zu jeder STD-From eine (vorhergehende) LNG-Form und zu jeder PRT-Form (vorhergehende) STD- und PRT-Formen. Trotzdem ist es möglich, dass in der Datenbank Einträge vorhanden sind, wo frühere Formen fehlen (z.B. nur PRT, ohne STD und LNG oder PRT, ohne STD aber mit LNG). In diesem Fall verwendet VSTENO einfach die in der Abarbeitungsreihenfolge letzte Form (hier also PRT).

Verzweigungen

Funktionen können auch definieren, welche Regeln bzw. welche Funktion als nächstes ausgeführt werden soll. Wird nichts angegeben, wird einfach die nächste Regel (aus der folgenden Funktion) ausgeführt. Alternativ kann eine bedingte Verzweigung angegeben werden:

```
#BeginSubSection(trickster)[92]
     // Regeln
#EndSubSection(trickster,=>decapitalizer,!>filter)
```

Diese Funktion definiert folgende bedingte Verzweigungen:

1. Wenn sich das Wort nach Ausführung des Trickster nicht geändert hat, verzweige zu Funktion decapitalizer.
2. Wenn sich das Wort nach Ausführung von Trickster verändert hat, verzweige zu Funktion filter.

Verglichen wird also das Wort am Anfang und am Ende der Funktion. Bedingte und unbedingte Verzweigungen können in VSTENO mit folgenden Zeichen definiert werden:

- =>: "verzweige wenn gleich"
- !>: "verzweige wenn nicht gleich"
- >>: "verzweige in jedem Fall"

Gross- und Kleinbuchstaben

Dieses Thema haben wir bis jetzt stillschweigend übergangen bzw. nur hie und da kurz angedeutet. Es geht um die Frage, wie Klein- und Grossbuchstaben in VSTENO verwendet werden.

Tatsache ist, dass Gross- und Kleinschreibung durchaus einen Unterschied machen können: das Substantiv "Waren" und das Verb "waren" werden in Stolze-Schrey z.B. völlig unterschiedlich geschrieben. Dies ist der Grund, warum VSTENO zu Beginn der ParserChain die Gross- und Kleinschreibung beibehält - und zwar bis und mit Shortener. Wenn Sie also im Shortener die Regel:

```
"war" => "{WAR}"
```

[92] Hier also nochmals der berüchtigte Trickster - es bleibt zu hoffen, dass VSTENO sich dieses unseligen Compagnons bis zur finalen Version 0.1 definitiv entledigen kann ... Der Trickster eignet sich deshalb als Beispiel, weil er gleich zwei bedingte Verzweigungen enthält. Nachtrag 08.04.2019: Der Trickster ist inzwischen Geschichte - um nicht die gesamte Dokumentation anpassen zu müssen, wird er hier als (nunmehr theoretisches) Beispiel belassen.

definieren, so wird diese nur auf das Verb "waren" (ergibt "{WAR}en") nicht aber auf "Waren" (bleibt "Waren") angewendet[93]. Da es bei den meisten Wörtern jedoch keinen Unterschied macht, ob sie gross oder klein geschrieben werden - und es mühsam wäre, in den Regeln jedesmal sämtliche Varianten in eckigen Klammern anzugeben - wandelt VSTENO nach dem Shortener sämtliche Gross- in Kleinbuchstaben um! Alle nachfolgenden Schritte (Normalizer, Bundler, Transcriptor, Substituter) arbeiten somit mit Kleinbuchstaben. Ausgenommen von der Umwandlung in Kleinbuchstaben sind allerdings Veränderungen, die der Shortener selbst vornimmt: Wenn der Shortener im Wort "wahrhaftig" also die Kürzung -haft zu "wahr{HAFT}ig" ersetzt, so verbleibt der Teil "{HAFT}" auch nach dem Shortener in Grossbuchstaben.[94]

Die konsequente Kleinschreibung - mit Ausnahme der vom Shortener eingesetzten Teile, wie gesagt - hat den Vorteil, dass wir Grossbuchstaben nun dazu benutzen können, jene Teile des Wortes zu markieren, die von bestimmten Regeln bereits umgeschrieben wurden: ein Kleinbuchstabe bedeutet also "wurde noch nicht bearbeitet" (= "muss bearbeitet werden") ein Grossbuchstabe bedeutet "wurde bereits bearbeitet" (= "darf nicht noch einmal bearbeitet werden")[95].

VPAINT

Wie bereits erwähnt wurde VPAINT als Hilfsprogramm für VSTENO entwickelt: In der Version SE1 rev0 ist die Definition der Zeichen grundsätzlich nur via einen Text-Editor möglich. Stenografische Zeichen erscheinen darin als lange Listen abstrakter Zahlen, die keine visuelle Vorstellung der Zeichen erlaubt und ein intuitives Entwerfen und Bearbeiten von Stenozeichen praktisch unmöglich machen. Als Abhilfe sollte hier der grafische Editor VPAINT (VPAINT - an Amazing Interactive New Tool[96]) schaffen.

Wie bereits in Version (Rückblick) dargelegt, war eigentlich geplant, mit VPAINT "durchzustarten", d.h. mit VPAINT eine neue (bessere) SE2 zu entwickelt, diese sowohl in JavaScript als auch in PHP zu implementieren und schliesslich die SE1 in einer "Operation am offenen Herzen" vollständig aus VSTENO zu entfernen und durch die SE2 zu ersetzen. Aus bereits dargelegten Gründen wurden diese Pläne geändert und stattdessen versucht, VPAINT mit einer - zwischenzeitlich zu einer rev1 weiterentwickelten SE1 - rückwärtskompatibel zu machen. Dies alles gefolgt von der weiteren Entscheidung, in der Version 0.1rc die SE1 rev1 komplett zu deaktivieren und nur dei SE1 rev0 zu verwenden.

[93] Eine andere Sache ist natürlich, wenn das Verb "Waren" dummerweise am Satzanfang steht und ebenfalls gross geschrieben wird ... ;-)

[94] Wir weisen hier - en passant - noch auf einen weiteren Aspekt hin: Wichtig ist in diesem Beispiel auch, dass der Normalizer (der den Vokal "ah" zu "a" umschreibt) erst NACH dem Shortener abgearbeitet wird, da sonst die Kürzung "war" auch auf das Wort "wa(h)rhaftig" angewandt würde.

[95] Die Umwandlung von Gross- zu Kleinbuchstaben (mittels strotlower() im consequence-Teil der Regeln) zu genau diesem Zweck erfolgt inzwischen mehrmals innerhalb des Modells Stolze-Schrey (siehe hier: https://github.com/marcelmaci/vsteno/blob/master/ling/grundschrift_stolze_schrey_redesign.txt).

[96] Vorläufig aber eher VPAINT - an Awfully Instable Nerd Tool (und man möge mir zugestehen, dass im Englischen die Variante instable für unstable, wenn auch seltener, durchaus existiert;-)

Der langen Rede kurzer Sinn: VPAINT ist mit der SE1 rev0 inkompatibel! Dennoch kann VPAINT in der Version 0.1rc von VSTENO wenigstens dazu verwendet werden, Stenografiezeichen zu visualisieren und - in eingeschränktem Masse - zu editieren, um schönere Zeichen zu gestalten und das Entwerfen der Stenozeichen intuitiver zu gestalten.

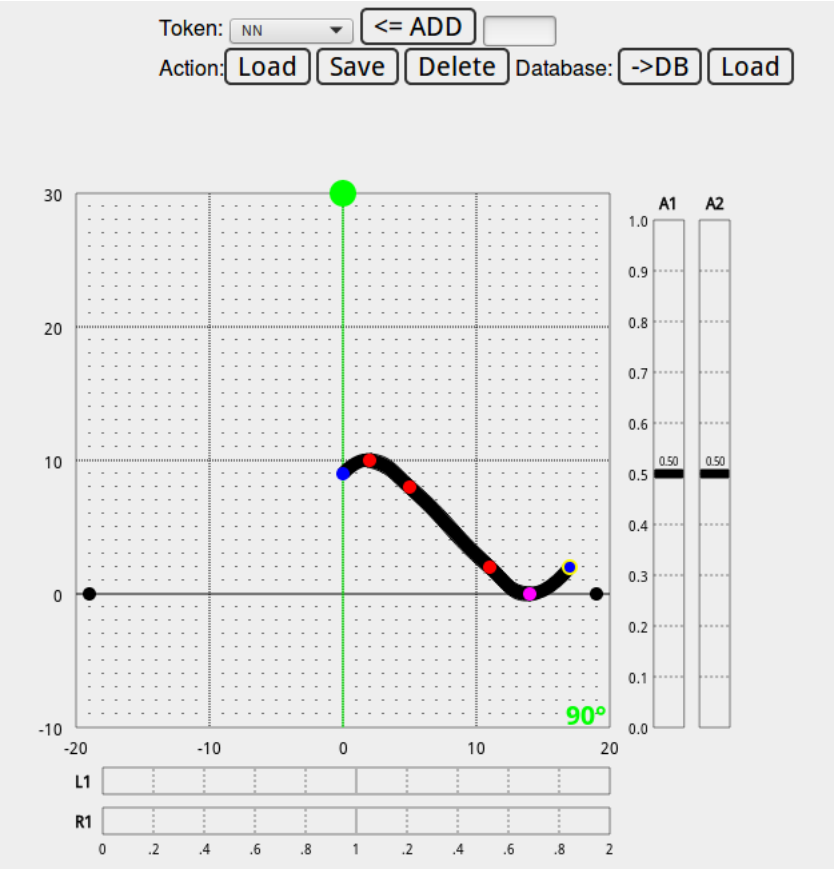
Die folgenden Ausführen sollen somit aufzeigen, was möglich ist - und was Sie tunlichst (!) vermeiden sollten!

Visualisieren

Mit VPAINT können Stenozeichen aus dem Text-Editor (also so, wie wir sie bis jetzt editiert haben) visualisiert werden. Gehen Sie hierzu wie folgt vor:

1. Gehen Sie auf die Seite www.vsteno.ch und loggen Sie sich ein.
2. Wählen Sie das Modell custom (1x links unten auf den Button standard klicken)
3. Klicken Sie in der linken Navigationsleiste auf VPAINT (das Programm wird geöffnet)
4. Laden Sie nun ein Zeichen, indem Sie aus dem Pulldown-Menü Select das Zeichen wählen und auf Load klicken.

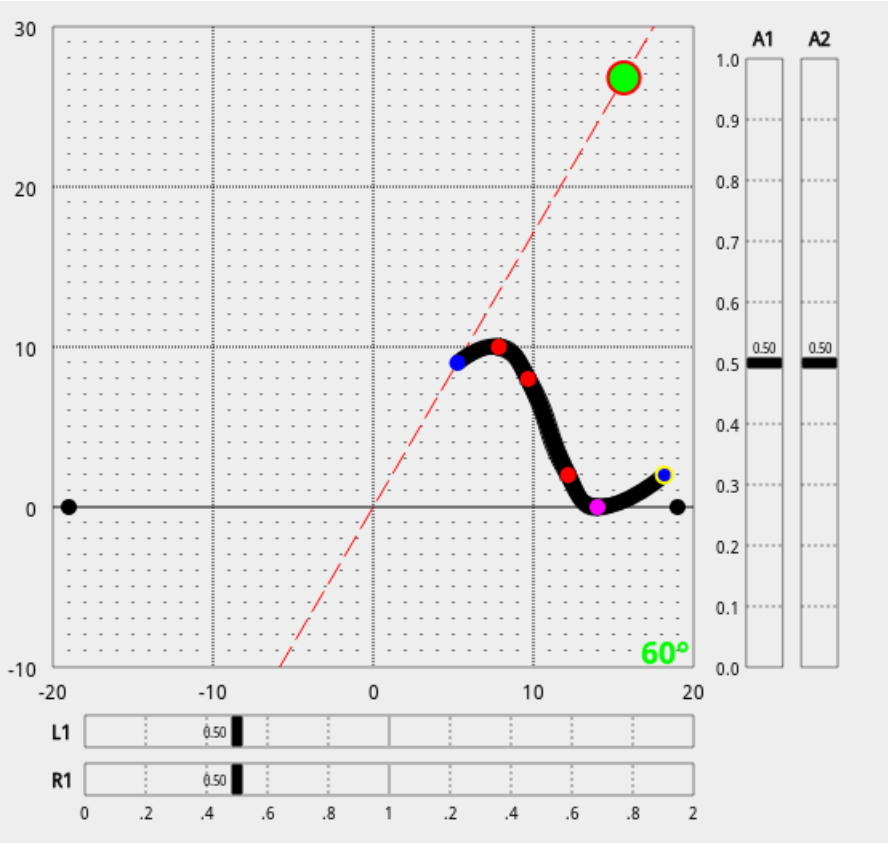
Sie sehen nun die folgende Darstellung:



Die Darstellung zeigt das Zeichen "NN" aus dem System Stolze-Schrey in vertikaler Stellung (90 Grad).

Editieren

Sie können das geladene Zeichen nun editieren. Da die SE1 rev0 nur horizontale Punktverschiebungen beherrscht, empfiehlt sich das folgende Vorgehen: Neigen Sie die Rotationsachse auf die gewünschte Neigung (z.B. 60 Grad in der Standardeinstellung von VSTENO). Sie erreichen dies durch Ziehen (klicken und ziehen) des grünen Punktes der Rotationsachse mit der Maus.



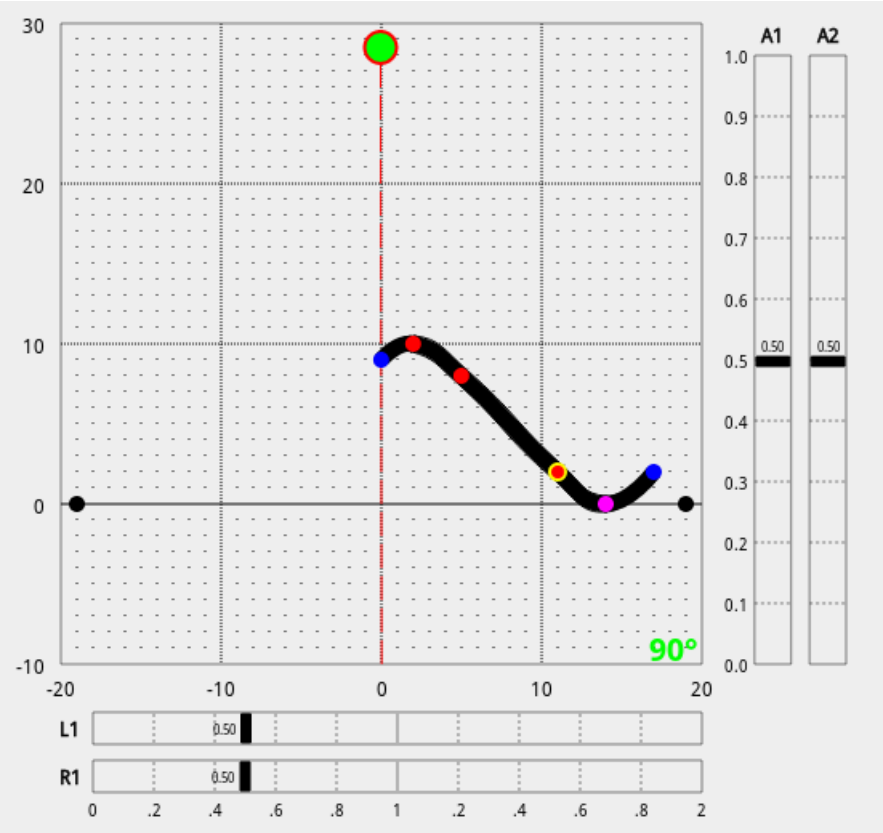
Editieren Sie nun das Zeichen so, dass es in dieser Neigung gut aussieht:

- Verschieben der Punkte mit der Maus (der aktuelle Punkt ist gelb umrandet).
- Anpassen der Spannungen (A1 = Eingangsspannung, A2 = Ausgangsspannung).

Mit der Taste "s" können Sie zwischen der normalen und der schattierten Variante des Zeichens hin- und herwechseln. Die Strichdicke kann mit L1 und R1 angepasst werden[97].

Wenn das Zeichen in der 60-Grad-Stellung gut aussieht, richten Sie die Rotationsachse wieder auf 90 Grad aus.

[97] Im Unterschied zur SE1 verfügt die SE2 über eine linke (L1) und eine rechte Dicke (R1). Die Dicke 1 (L1, R1) steht für das normale, die Dicke 2 (L2, R2) für das schattierte Zeichen. Beachten Sie bitte, dass die Dicke der SE1 dem Mittelwert von L1 und L1 (L2 und L2 für das schattierte Zeichen) entspricht, als TH = (L1+L2) / 2. Um die linke und rechte Dicke der SE2 zu visualisieren, können Sie mit der Taste "q" zwischen SE1 (Mittellinienmodellierung) und SE2 (Umrissmodellierung) umschalten.



Und nun kommt der schöne Teil: Nehmen Sie ein Blatt Papier und notieren Sie sich die Koordinaten und die Spannungen (und eventuell Dicken) der Punkte der Reihe nach[98]. Ja genau: Es ist Handarbeit gefragt! VPAINT ist mit der SE1 rev0 inkompatibel - versuchen Sie nicht die Daten zu speichern - klicken Sie NICHT auf den Button ->DB![99]

VPAINT kann Ihnen also lediglich helfen, bestehende Zeichen zu visualisieren und zu editieren. Die Daten müssen anschliessend aber von Hand via Texteditor in VSTENO 0.1rc (SE1 rev0) übertragen werden.

[98] Sie können sich mit den Pfeiltasten < und > rechts jeweils einen Punkt nach links oder nach rechts bewegen.

[99] Im Hinblick auf die Release der Version 0.1 wurde die Speicherfunktion von VPAINT nun definitiv deaktiviert. Es besteht somit keine Gefahr mehr, Daten und Modelle durch fehlerhafte Exporte zu beschädigen. Wenn auf den Button ->DB geklickt wird, gelangt man wie gewohnt zu einem Texteditor, der die exportierten Zeichen-Definition enthält. Daraus können - wenn gewünscht - einzelne Zeichen manuell kopiert und via den normalen Font-Editor (aus der Seitennavigation) in ein bestimmtes Modell eingefügt werden. Wenn Sie anschliessend auf "speichern" klicken, so führt der Link lediglich auf die Seite "edit_font_disabled.php", wo Sie darauf hingewiesen werden, dass die Speicher-Funktion deaktiviert wurde.

Zukunftsmusik

Wenn Sie gerne einen Blick auf die SE2 werfen und etwas mit dem Programm VPAINT herumprobieren möchten, dann verwenden Sie dafür bitte den folgenden Link:

https://www.vsteno.ch/js/vsteno_editor.html

Dies ist die allgemeine (d.h. nicht an VSTENO gekoppelte) Version von VPAINT. Sie funktioniert auch, wenn Sie nicht eingeloggt sind und Sie können hier also sicher sein, dass Sie nichts kaputt machen können. Sie finden hier auch eine Auflistung der Tastaturbefehlen, mit denen Sie weitere Funktionen von VPAINT nutzen können.

Programm

Ja, ja ... Dokumentation

Nach dem praktischen und dem linguistischen Teil stünde nun natürlich noch die Dokumentation des (bzw. der) Programme(s) selber an. Dies, damit interessierte Programmierer/innen VSTENO unter Umständen klonen (oder forken:) und selber weiterentwickeln können.

Wie allgemein bekannt ist das Dokumentieren so ein bisschen das Stiefkind des/der Programmierer/in: Es wird in der Regel erst ganz am Schluss erledigt - wenn das Progamm (endlich!) läuft und man weder Zeit noch wirklich Lust hat, "das alles auch noch aufzuschreiben". Ausserdem kann es gut sein, dass gewisse Programmteile bereits vor Monaten geschrieben wurden - und nun, wo man sie dokumentieren sollte, weiss man selber nicht mehr so genau, was man damals eigentlich so alles programmiert hat, und muss sich deshalb selber wieder in alten Code reinknien (von dem man, wie gesagt, überhaupt schon froh ist, dass er läuft:)

Anyway: Ich bin keine Ausnahme von der Regel und berufe mich deshalb auf eine weitere Programmierregel: my code is my documentation ... ! Natürlich ist das die Billig-Lösung, aber sie ist so schlecht nicht: Tatsächlich habe ich im Laufe der Entwicklung von VSTENO den Quellcode reichlich dokumentiert. Anhand der eingefügten Kommentare sollte das Programm also grundsätzlich nachvollziehbar sein.

Nicht nachzuvollziehen sind aber vielleicht gewisse allgemeine Entscheidungen und Ansätze, die bei diesem Projekt spezifisch sind. Auf einige solche Aspekte möchte ich im Folgenden doch kurz eingehen, um wenigstens den Zugang zum Programm zu erleichtern.

Ebenfalls wichtig scheint mir zu dokumentieren, wie das Programm grundsätzlich lokal installiert und zum Laufen gebracht werden kann. Es existiert bis dato nämlich kein Installer und VSTENO verlangt einige Abhängigkeiten und Konfigurationen, die korrekt vorgenommen werden müssen, damit es läuft.

Grundsätzlich werde ich mich in allen Ausführungen sehr kurz und allgemein halten und als erstes auch nur das Minimum dokumentieren[100] und weitere Aspekt

[100] Weil es ja seit August schon eine Weile her ist, dass das Programm das letzte Mal dokumentiert

später ergänzen.

Installieren[101]

Wenn Sie VSTENO lokal installieren möchten, müssen einige Pakete (Abhängikeiten) installiert werden und die Datenbank vorgängig konfiguriert werden. Im folgenden gehe ich davon aus, dass Sie grundsätzlich mit der Installation von Paketen vertraut sind (wie z.B. sudo apt-get install <paketname> unter Debian[102] oder anderen Paketmanagern auf anderen Disktributionen). Ich werde mich deshalb auf die Angabe der Pakete beschränken, die installiert werden müssen:

1. Apache-Webserve
2. PHP
3. mySQL (ebenfalls nützlich: MySQL Workbench)
4. hunspell

Klonen Sie anschliessend die Programme VSTENO und VPAINT (sowie verwendete Libraries wie paper.js und phpSyllable):

1. git clone https://github.com/marcelmaci/vsteno[103]

Konfigurieren Sie die Datenbank:

1. Legen Sie einen Datenbankbenutzer an
2. Tragen Sie die Zugangsdaten in den PHP-Quellcode ein (Datei dbpw.php)
3. Erstellen Sie die leeren Datanbank-Tables (kann einfach erreicht werden, indem die Datei init_db.php im Browser aufgerufen wird)

Wenn Sie nun localhost/vsteno/php/input.php im Browser aufrufen, sollten Sie das Eingabe-Formular von VSTENO sehen. Wenn dies der Fall ist, läuft das Programm - allerdings enhält es noch keine linguistischen Daten.

1. Legen Sie ein VSTENO-Benutzerkonto an (leer) und wählen Sie das Modell custom.
2. Holen Sie sich die Definitionen für das System Stolze-Schrey [104].
3. Kopieren Sie Header, Font und Rules in die Datenbank (nutzen Sie hierfür die Links zu den Texteditoren: Header, Zeichen, Regeln).

---

wurde. Im Moment soll die Dokumentation vor allem möglichst schnell verfügbar sein.

[101] Die folgenden Ausführungen geben nur einen groben Überblick. Für eine sehr detaillierte Anleitung kann das Dokument https://github.com/marcelmaci/vsteno/blob/master/docs/install_vsteno.txt konsultiert werden.

[102] VSTENO wurde unter Trisquel, einem Debian-Abkömmling, entwickelt.

[103] Im Verzeichnis des Webservers (vermutlich /var/www/html).

[104] https://github.com/marcelmaci/vsteno/blob/master/ling/grundschrift_stolze_schrey_redesign.txt

4. Loggen Sie sich mit einem Datenbankprogramm (mySQL Workbench) in die Datenbank ein:
   (a) öffnen Sie die Table models
   (b) tragen Sie folgende Werte ein: user_id = 99999, name = DESSBAS

Loggen Sie sich aus, schliessen Sie den Browser und starten Sie das Ganze neu. Mit etwas Glück funktioniert's ... ! ;-)

VSTENO

Im Folgenden einige Hinweise zu VSTENO:

- VSTENO wurde ausschliesslich in PHP (ohne JavaScript[105]) programmiert.
- Das Verzeichnis php/ enthält nicht nur den Quellcode sondern auch die Webseite von VSTENO[106].
- Sämtlicher Quellcode von VSTENO befindet sich innerhalb des Verzeichnisses php/.
- VSTENO verwendet die Bibliothek phpSyllable, die sich im Verzeichnis ../phpSyllable/ befindet.

Alles Übrige sollte ziemlich selbsterklärend sein.

VPAINT

- VPAINT wurde in PaperScript programmiert, d.h. JavaScript unter Verwendung der Bibliothek paper.js.
- Sämtlicher Code (auch der von paperl.js) befindet sich im Verzeichnis ../js/
- Das Skript buildjs.sh fügt alle Programmteile von VPAINT zu einem einzigen File zusammen[107].

Auch hier sollte der Rest selbsterklärend sein.

JSON & Co

VSTENO ist eine fröhliche Import-Export-Escape-Hölle ... :-) Mit anderen Worten: Da zwei Programmiersprachen, eine Datenbank, HTML-Code, REGEX und auch die Shell verwendet wird, müssen die Daten zum Teil mehrmals von einer Umgebung in eine andere überführt, konvertiert und escaped werden.

Während das Escaping mit Standard-Funktionen erledigt werden kann, war der export der linguistischen Daten von PHP zu JS etwas speziell und soll deshalb hier kurz erläutert werden:

[105] Einzige Ausnahme ist die Deaktivierung des Tabulator-Events für den Texteditor
[106] Wer also nur den Programmcode möchte, muss diesen von der Webseite trennen.
[107] Ich wollte den Quellcode einigermassen überschaubar aufteilen und die Möglichkeit haben, Lizenzvermerke automatisiert einzufügen, deshalb die Aufteilung und das Skript.

- An Anfang lagen die Daten von VSTENO innerhalb von PHP als (associative) arrays vor.
- VPAINT definierte seine eigenen Datenstrukturen[108].
- Mit der Idee der Rückwärtskompatibilität wurde ein Export von PHP zu JS realisiert:
  - in PHP wird eine mit JS kompatible Objekt-Datenstruktur definiert[109].
  - die Datenstruktur wird mit JSON exportiert[110].
- Der umgekehrte Weg, also von JS zu PHP, sollte in der SE1 rev1 folgendermassen laufen:
  - VPAINT exportiert die Daten ins eigenen VSTENO-Format (ASCII-Text, der vom Parser von VSTENO geparst werden kann).
  - Um die Datenbank-Routinen nicht nochmals neu schreiben zu müssen, kreiiert VPAINT eine Formular-Seite, die genau gleich aussieht wie edit_font.php und die zu schreibenden Daten direkt enthält).
  - Von der Datenbank aus kann VSTENO die Daten wieder lesen und parsen, sodass sie wieder im ursprünglichen Format der (associative) arrays vorliegen.

Der Datenfluss von VSTENO zu VPAINT und zurück zu VSTENO ist also gewissermassen zirkulär: er beschreitet andere Wege und kann auch nicht umgekehrt werden (JSON-Export funktioniert nur von PHP->JS; ASCII-Export funktioniert nur von JS->DB->PHP; unmöglich ist also JSON: JS->PHP oder ASCII: DB->JS).

Für die SE2 war (ist) vorgesehen sämtliche Import/Export von und zu der Datenbank sowohl von PHP als auch von JS aus mit JSON zu lösen.

PARSER

Der ASCII-Parser von VSTENO (von dem im vorherigen Kapitel die Rede war) befindet sich in parser.php und ist im Wesentlichen eine Abfolge von REGEX-Pattern, mit denen die ASCII-Daten aufgeschlüsselt und in Variablen geschrieben werden.

Für den Export der Daten von PHP zur Datenbank werden die Daten dann gemäss der Formelsyntax von VSTENO zusammengefügt.

DATENBANK

Die Datenbank enthält folgende Tables:

- users: Benutzerdaten (login, passwort, name, email etc.)

[108] Es war ja vorgesehen, die SE1 zu entfernen und die SE2 komplett neu zu gestalten.
[109] siehe export_se1_data_to_editor.php.
[110] Das PHP-Objekt wird "stringified" eine JS-Variablen zugwiesen, die direkt in die HTML-Seite von VPAINT geschrieben wird; von dort verwendet Sie VPAINT wie eine normale Variable.

- models: Definitionen für Stenografische Systeme (header, font, rules; die Modelle haben einen Namen und sind mit einer user_id verknüpft)
- Wörterbücher:
  - purgatorium: enthält Wörter, die markiert wurden und reviewed werden müssen.
  - elysium: enthält Ausnahmen (STD-, PRT-Formen)
  - olympus: enthält regelmässige, richtige Wörter (die zum Überprüfen des Modells verwendet werden können)

Die Tables users und models werden nur ein Mal für alle Nutzer angeleget. Die Tables purgatorium, elysium und olympus werden für jeden Nutzer individuell angelegt. Der Table name ergibt sich dabei aus der user_id der Benutzers plus den Anfangsbuchstaben der Table:

- E = elysium
- P = purgatorium
- O = olympus

Ausserdem wir zwischen einem Standard und einem Custom-Modell unterschieden:

- Z = standard
- X = custom

Hier ein paar Beispiele für Table-Namen:

- XE0000008: custom Wörterbuch Elysium der user_id 8
- ZP0000001: standard Wörterbuch Purgatorium der user_id 1

SUPERUSER

In VSTENO gibt es Standard-Nutzer (privilege level 1) und Superuser (privilege level 2). Standard-Nutzer können nur custom Wörterbücher beschreiben. Superuser können alle Tables beschreiben. VSTENO bietet keine erweiterten Funktionen zur Nutzerverwaltung: Nutzer können nur 1x angelegt werden. Danach kann das Passwort und andere Nutzerdaten nicht mehr geändert werden. Solche Änderungen (z.B. die Erhöhung des privilege levels müssen deshalb von Hand in der Datenbank vorgenommen werden).

SESSION

Die Session-Variable wird von VSTENO rege genutzt. Sämtliche benutzten Daten sind aus session.php ersichtlich. Die Variable können auch via Inline-Option-Tags verändert werden[111].

[111] https://www.vsteno.ch/docs/vsteno_tutorial.pdf

Lizenzen & Copyrights

VSTENO wird unter der GPL-License veröffentlicht und ist somit Freie Software. Das Programm darf also kopiert, verändert, weiterverbreitet und auch kommerziell genutzt werden, sofern die Bedingungen der GPL-License beachtet werden. Hierzu gehört insbesondere das Copyleft: Wird das Programm verändert und kommerziell genutzt, so muss auch das veränderte Programm als Freie Software unter der GPL herausgegeben und dessen Quellcode veröffentlicht werden. Damit soll verhindert werden, dass VSTENO durch irgendjemanden "vereinnahmt" und "unfrei" gemacht werden kann. VSTENO soll auch in Zukunft frei bleiben und jegliche Entwicklung an VSTENO soll dem ursprünglichen Projekt vollumfänglich zugute kommen.

Im Folgenden werden nun die für VSTENO gültigen Lizenzen in ihrer Kurzfassung publiziert. Nebst den allgemeinen Bestimmungen der GPL sind im Falle von VSTENO einige zusätzliche Präzisierungen nötig. Diese betreffen stenografische Modelle, Datenbanken, Dokumentationen, zusätzliche verwendete Programme und Bibliotheken und schliesslich auch Dokumente (Erzeugnisse), die mit VSTENO erstellt werden.

Grundlizenz (GPL)

Die GPL bildet die Grundlizenz von VSTENO. Sie ist für den selbst entwickelten Kernteil des Programmes, aber auch für das Gesamtprojekt (mit linguistischen Modellen, Datenbanken und Bibliotheken) gültig:

> VSTENO - Vector Steno Tool with Enhanced Notational Options
> (c) 2018-2019 - Marcel Maci (m.maci@gmx.ch)
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any laterversion.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Dies Grundlizenz umfasst sämtliche Programmteile von VSTENO, also insbesondere auch VPAINT: Wann immer von VSTENO die Rede ist, so ist VPAINT mitgemeint.

Modelle

Stenografische Modelle sind eine formalisierte, strukturierte Sammlung (Sections/SubSections, Wenn-Dann-Konstrukte, Verzweigungen, Zuweisen von Variablen etc.) von Regeln und Anweisungen, mithilfe derer ein stenografisches System umgesetzt werden kann. VSTENO fungiert in diesem Fall als Interpreter, der diese Regeln und Anweisungen abarbeitet. Mit anderen Worten: Modelle können selber als "Programm" betrachtet

werden und werden somit nicht als "Daten" von VSTENO, sondern als "Teilprogramme" von VSTENO behandelt, die ebenfalls unter die (gleiche) GPL fallen.

Dies bedeutet, dass sämtliche Modelle, die mit VSTENO (neu) erstellt oder (in geänderter oder unveränderter Form) verwendet werden, ebenfalls unter der GPL und mit den entsprechenden Hinweisen (Veröffentlichung der Lizenz und Angabe des/der Autor/in) publiziert werden müssen. Die Auslegung des Copylefts ist in diesem Punkt also sehr strikt: Sämtliche Stenografie-Systeme, die mit VSTENO umgesetzt werden, müssen ihrerseits frei sein. Diese Ergänzung betrifft die öffentliche (u.U. auch kommerzielle) Nutzung solcher Modelle (die private Nutzung und Umsetzung von Stenografie-Systemen ist frei).

Datenbank

Die Datenbanken von VSTENO enthalten einerseits Nutzer/innen-Daten (Passwort, Login etc.), andererseits Daten, die in Zusammenhang mit den Modellen stehen ("unregelmässige" Wörter, die speziell abgearbeitet werden müssen).

Nutzer/innen-Daten sind von den folgenden Einschränkungen ausgenommen: Diese Daten unterliegen dem Datenschutz. Sie müssen (und sollen) nicht publiziert werden!

Die Daten, welche im Zusammenhang mit den verwendeten Modell stehen (insbesondere aus Olympus, Elysium, Purgatorium) werden analog zu den Modellen nicht als "Daten" sondern als "Teil des Programmes" betrachtet: Tatsächlich kann jeder Datenbankeintrag als zusätzliche Regel eines Modells verstanden werden (Wenn-Dann-Konstrukt mit ganzen Wörtern statt Wortteilen). In diesem Sinne unterliegen auch diese Teile der Datenbank der GPL und müssen zusammen mit dem Modell publiziert werden.

Dokumentationen

Sämliche Dokumentationen (PDF, HTML[112], TXT) von VSTENO unterliegen der Free Documentation License (FDL[113]):

> Copyright (C) 2018-2019 Marcel Maci
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Bibliotheken

VSTENO verwendet zwei Bibliotheken: (1) paper.js (VPAINT) und (2) phpSyllable (VSTENO). Beide verwenden die MIT-Lizenz, die mit der GPL kompatibel ist. Im

[112] Webseite, was nicht Teil des Programmes ist.
[113] https://static.fsf.org/nosvn/directory/fdl-1.3-standalone.html

Folgenden werden die Lizenzen im originalen Wortlaut wiedergegeben.

PAPER.JS

phpSyllable

> WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Die MIT-Lizenz ist eine so genannte "permissive" Lizenz (ohne Copyleft), die mit der GPL kompatibel ist - jedoch nicht umgekehrt. Aus diesem Grund fallen diese Programmteile im Zusammenhang mit VSTENO ebenfalls (zusätzlich) unter die GPL.

hunspell

Hunspell[114] wird von VSTENO zur linguistischen Analyse genutzt. Hierzu muss hunspell bereits auf dem System installiert sein. Der Aufruf erfolgt via Shell. Aus diesem Grund ist hunspell nicht Teil von VSTENO.

eSpeak

eSpeak[115] wird von VSTENO zur phonetischen Transkription genutzt. Hierzu muss eSpeak auf dem System installiert sein. Der Aufruf erfolgt via Shell. Aus diesem Grund ist eSpeak nicht Teil von VSTENO.

Erzeugnisse

Erzeugnisse sind Dokumente, die mit VSTENO und einem entsprechenden Modell & Datenbank erstellt werden (z.B. Stenografie-Lektüren). Die Erstellung und Nutzung solcher Erzeugnisse zu privaten Zwecken ist frei und an keine Einschränkungen gebunden. Im Fall einer öffentlichen Nutzung müssen sämtliche Copyrights gewahrt werden, indem folgender Hinweis an einer beliebigen Stelle im Dokument (z.B. Impressum) erscheint:

```
Erstellt mit Freier Software.
VSTENO (www.vsteno.ch)
```

Der Hinweis muss zusätzlich versehen sein mit: Copyrightvermerk, Jahr und Namen des/der Autors/en sowohl des Programmes als auch des Modells. Weitere nützliche Angaben (wie Programmversion und genaue Angabe des verwendeten Modells), die es dem/der Leser/in erleichtern, transparent nachzuvollziehen, wie das Dokument erstellt wurden, sollen - wenn möglich - ebenfalls aufgeführt werden. Gleiche Angaben können zusammengefasst werden, damit eine kompaktere und ästhetischere Darstellung möglich ist, z.B.

```
Erstellt mit Freier Software.
VSTENO (www.vsteno.ch)
(c) 2018-2019 Marcel Maci
```

[114] https://hunspell.github.io/
[115] http://espeak.sourceforge.net/

als Kurzfassung oder

```
Erstellt mit Freier Software.
VSTENO 0.1 (www.vsteno.ch)
Stolze-Schrey Grundschrift (DESSBAS)
(c) 2018-2019 Marcel Maci
```

mit zusätzlichen Angaben zur Programmversion und zum verwendeten Modell.

Lizenzkonflikte und Relizenzierung

Die Lizenzierung unter der GPL erfolgt hauptsächlich aus zwei Gründen: (1) es ist sichergestellt, dass Programmcode (und weitere Teile von VSTENO wie stenografische Modelle etc.) grundsätzlich frei verwendet werden können und (2) es wird ausgeschlossen, dass VSTENO von irgend jemandem "vereinnahmt" (d.h. unfrei gemacht) und zum Beispiel in kommerziellen, proprietären Produkten verwendet werden kann, ohne allfällige weitere Entwicklungsarbeit daran (und insbesondere den Quellcode) offen zu legen.

In den meisten Fällen bedeutet dies für freie Software-Lizenzen ohne Copyleft (z.B. MIT, BSD), dass nur dann Teile aus VSTENO übernommen können, wenn das daraus entstehende Projekt inskünftig unter der GPL herausgegeben wird (weil besagte Lizenzen aufgrund des nicht vorhandenen Copylefts den obgenannten Punkt 2 missachten bzw. nicht sicherstellen). Dies mag für die Anhänger/innen dieser Lizenzen (die im übrigen durchaus ihre Berechtigungen haben) zwar bedauerlich sein, dennoch halte ich als Autor von VSTENO ganz klar am Copyleft fest. Ausserdem erachte ich das Problem als nicht so gravierend, da aufgrund der (wenngleich nur einseitigen) Kompatibilität der Lizenzen hier immerhin die Möglichkeit der Lizenzierung unter der GPL besteht, um ein entsprechendes Projekt weiterzuführen (wodurch also der Punkt 1 - wenn auch in etwas eingeschränkterer Form - sichergestellt ist).

In anderen Fällen ist die GPL - als sehr eng gefasste Lizenz - jedoch grundsätzlich inkompatibel, d.h. es besteht keine Möglichkeit, Code aus zwei (oder mehreren) Projekten zusammenzuführen und daraus ein neues Projekt mit einer konfliktfreien Lizenz zu erstellen. Dies trifft konkret auf die APL (AROS Public License) zu, die für das freie Betriebssystem AROS (eine Nachprogrammierung von AmigaOS auf Amiga Computern) verwendet wird. Als grosser Amiga-Fan - der zu Weihnachten 1987 einen Amiga 500 geschenkt bekam und von der Begeisterung für diese Computer bis heute nicht losgekommen ist[116] - ist mir dies ein Dorn im Auge. Gerade der Fall Amiga hat überdeutlich gezeigt, dass einschränkende und proprietäre Lizenzen, verbunden mit den daraus folgenden Rechtsstreitigkeiten und der Unmöglichkeit, bestehende Software zu übernehmen und weiterzuentwickeln, massgeblich mitverantwortlich waren und sind für den Untergang und die Blockierung dieser Plattform seit spätestens 1996! Freie Software sollte gerade das Gegenteil sein, d.h. den freien

[116] zumal voraussichtlich 2019 mit der so genannten Vampire V4 Standalone erstmals nach über 25 Jahren wieder ein wirklich als solcher zu bezeichnender, "neuer Amiga" herauskommt, siehe hier: https://www.apollo-accelerators.com/

Austausch ermöglichen. Dies haben auch die Gründer von AROS erkannt und die APL unter Punkt 3.2.[117] mit dem Hinweis versehen, dass der Quellcode zur Verfügung gestellt werden muss. Da dies das Copyleft aus Punkt 2 in seinem wichtigsten Punkt (und somit ausreichend) wahrt, ist die Inkompatibilität von GPL und APL ein Stolperstein, der aus meiner Sicht weder sein soll, noch muss.

In diesem Sinne erhält VSTENO diesen zusätzlichen Relizensierungs-Hinweis: Sämtliche Teile von VSTENO dürfen in ein unter der APL stehendes Projekt übernommen und unter der APL relizensiert werden. Das Relizensierungsrecht wird jedoch nur eingeschränkt verliehen: (1) Einerseits gilt es nur für die einmalige Relizenzierung unter der APL, nicht jedoch für den Fall, dass ein unter der APL stehendes und Teile von VSTENO enthaltendes Projekt ein weiteres Mal unter einer anderen Lizenz herausgegeben wird[118] und (2) andererseits, gilt die Relizensierung ausschliesslich für die APL (und ausdrücklich nicht für andere Software-Lizenzen). [119]. Das Relizensierungsrecht ist unabhängig von der Prozessorarchitektur (m68k, x86, ppc etc.), auf denen AROS läuft (und gilt somit für sämtliche aktuellen und zukünftigen Architekturen). Im Sinne der Einfachheit, wird dieses Relizensierungsrecht nur an dieser Stelle (und nicht in sämtlichen Projektteilen) aufgeführt.

Schluss

Diese Dokumentation entstand wie immer mit einem relativ knappen Zeitbudget. Ich hoffe, dass trotzdem die wesentlichsten Aspekte abgedeckt werden konnten. Bei Unklarheiten stehe ich gerne zur Verfügung (m.maci@gmx.ch).

[117] http://aros.sourceforge.net/license.html

[118] Soweit ich verstehe, ist die Relizenzierung von APL-Code unter einer anderen Lizenz grundsätzlich nicht möglich (ausser die Autoren stimmen dem zu), dennoch soll diese Einschränkung hier sicherheitshalber explizit festgehalten werden: Jegliche Relizenzierung unter einer Lizenz ohne Copyleft ist nicht in meinem Sinne. Ob man (ich) eine Relizenzierung erlauben aber gleichzeitig einschränken kann, ist eine rechtliche Frage, die mein Fassungsvermögen übersteigt. Deshalb hier einfach die generelle Anmerkungen, was bei einer Relizenzierung erlaubt sein soll und was nicht.

[119] Bei allfälligen Lizenzkonflikten besteht aber jederzeit die Möglichkeit, mich zu kontaktieren (m.maci@gmx.ch) und allfällige Zusatzklauseln zu vereinbaren, um die Konflikte bereinigen.