XIPA
Extensible International Phonetic Alphabet

Technische Dokumentation – Version 1.0

Sebastian Musche
5. März 2016

http://www.rhoticity.de/xipa/

INHALTSVERZEICHNIS

Die Konzeption von XIPA setzt auf die Entwicklung in Anlehnung an Standards und Empfehlungen des „World Wide Web Consortium“ (W3C), um eine langfristige Verwendbarkeit des Projektes durch die Nutzung zukunftstauglicher, nachhaltiger Technologien zu garantieren. Die Verwendung offener Standards sichert dabei die Wiederverwendbarkeit der Daten und die Austauschbarkeit der Daten über Plattformen hinweg.

Die Grundstruktur von XIPA beruht auf XML-basierten Datenformaten (XML, XSLT und SVG). Der Vorteil XML-basierter Daten liegt in der Möglichkeit einer verständlichen (logischen) Auszeichnung der Elemente sowie einer klaren Trennbarkeit von Form und Inhalt. Durch die Textform der Daten ist zur Bearbeitung der Dateien lediglich ein einfacher Texteditor notwendig.

Die interaktiven Elemente („Event-Handler“, z.B. zur Übertragung der Zeichen in das Textfeld, zur Ausgabe der Audiodateien und zur Anzeige des Tooltips) sind mit JavaScript realisiert. Einige Funktionen nutzen Methoden der JavaScript-Bibliothek jQuery. Alle verwendeten Technologien sind in aktuelle Browser implementiert.

Die Möglichkeit einer klaren Trennung von Inhalt und Form erlaubt es, dass Inhalte und Darstellungsoptionen unabhängig voneinander bearbeitet werden können. Sie ist damit eine wichtige Voraussetzung für die Weiterverarbeitung der Daten wie beispielsweise die Nutzung der gesammelten Informationen zu den phonetischen Zeichen in anderen Projekten oder die Auswertung mit anderen Werkzeugen.

1.1 XML-DATEN

Als Ausgangspunkt für die Zusammenstellung der einzelnen phonetischen Zeichen und der dazugehörigen Informationen wurde das Datenformat XML (Extensible Markup Language) gewählt. Den Kern des Projekts bildet eine ca. 5000 Zeilen Code umfassende XML-Datei.

Da XML – im Gegensatz zu HTML – keine festgelegte Liste von Auszeichnungselementen (Tags) zur Auszeichnung der Daten vorschreibt, ist eine Strukturierung der Daten durch selbstbenannte Tags möglich. Durch die Verwendung eines selbsterklärenden Markups wird die Lesbarkeit und Weiterverarbeitbarkeit der Daten erheblich erleichtert.

Allerdings enthält XML selbst keine Hinweise darüber, wie die Daten in einem Browser zu interpretieren sind. Die reine XML-Datei wird vom Browser lediglich als Baumstruktur – Quellcode mit Einrückung und farblicher Markierung – dargestellt. Um die Daten zu formatieren (d.h. in einer brauchbaren Form auszugeben) wird ein Stylesheet benötigt. Die Ausgabe kann entweder durch ein CSS-Stylesheet realisiert werden – sofern sich die Daten schon in einer geeigneten Reihenfolge befinden – oder durch ein XML-basiertes Stylesheet (z.B. eine XSL-Transformation durch ein XSLT-Stylesheet).

1.2 XSL-Transformation (XSLT)

XSLT dient der Umwandlung eines XML-Dokuments in ein anderes (XML-)Format. XSLT baut auf der abstrakten, durch das Markup repräsentierten Struktur des Quelldokuments auf, die einer Baumstruktur aus Knoten entspricht (siehe XPath). In XIPA wird das XML-Dokument mittels XSLT (Version 1.0) in das XML-konforme Format XHMTL konvertiert, d.h. in XML-gerechtes, vom Browser darstellbares HTML umgewandelt.

Das XSL-Stylesheet definiert dabei die Regeln, nach welchen die Baumstruktur des Quelldokuments in den Ergebnisbaum transformiert wird. Die Transformation bietet die Möglichkeit, Daten aus dem Quelldokument gezielt auszuwählen, die Inhalte einzelner Elemente zu kombinieren oder zu manipulieren und damit das resultierende Ergebnisdokument (bzw. dessen Formatierung und Gestaltung) an die gegebenen Bedürfnisse anzupassen.

Theoretisch kann die XSLT-Datei alle zur Ausgabe der HTML-Datei notwendigen Formatierungsangaben und alle Elemente zur optischen Gestaltung beinhalten. Um die Flexibilität zu erhalten, sollten Logik, Darstellung und Daten allerdings soweit wie möglich voneinander getrennt bleiben. Aus diesem Grund konzentriert sich die für XIPA generierte XSLT-Datei auf die Organisation der Elemente in Gruppen, auf die zur Formatierung HTML-Datei notwendigen Angaben und auf die Anreicherung der Elemente mit den für die Benutzerinteraktion notwendigen Attribut-Wert-Paaren.

1.3 XSL- und CSS-Stylesheets

Um Logik, Darstellung und Daten soweit wie möglich voneinander zu trennen, enthält die XSL-Datei möglichst wenig Gestaltungselemente. Lediglich einige Breitenangaben diverser Tabellenzellen sind in den einzelnen Templates hinterlegt. Die Ausgabe besteht also aus einem Dokument, dessen Formatierung bereits nahezu der endgültigen Darstellungsform entspricht. Die finale visuelle Gestaltung ist in XIPA in zwei weitere Dateien ausgegliedert – eine aus „Attribut-Sets“ bestehende XSL-Datei und eine CSS-Datei.

Innerhalb der XSL-Datei sind diverse „Attribut-Sets“ hinterlegt, die aus den einzelnen Template-Aufrufen des XSL-Stylesheets heraus aufgerufen werden. Die Attribut-Sets bestehen z.B. aus Angaben zu Rahmenlinien (für verbundene Zellenreihen), aus Breitenangaben für unterschiedliche Tabellenblöcke und weiteren „Style“-Elementen. Die Inhalte der innerhalb der Sets definierten Attribute (z.B. „style“ oder „colspan“) entsprechen der Notationsweise von CSS-Anweisungen. Sie bieten sich an denjenigen Stellen an, an welchen Attribut-Gruppen mehrmals wiederholt eingesetzt werden sollen, ohne dafür zusätzliche – durch CSS-Selektoren „ansprechbare“ – Attribute verwenden zu müssen.

Durch die Verwendung der XSL-Attribut-Sets sind nur noch wenige Elemente übrig, die separat mit (Styling-)Angaben versehen werden müssen. Diese letzten Angaben sind in eine CSS-Datei ausgelagert, die das grundsätzliche Aussehen der Grundelemente bestimmt und das finale Rendering im Browser bestimmt.

Innerhalb der CSS-Datei sind beispielsweise die Hintergrundfarbe von Tabellenzellen der Klasse „imp“ angegeben (um Laute die als nicht produzierbar klassifiziert sind hervorzuheben), das Verhalten bei Mausbewegungen über Symbole festgelegt (z.B Änderung des Mauszeigers beim Bewegen über „aktive“ Bereiche), sowie einige Angaben zur Ausgestaltung der Vektorgrafik und zum Erscheinungsbild des Tooltips gespeichert.

1.4 XPath (Navigation im XML-Strukturbaum)

XPath ist eine Sprache zur Adressierung von Teilen eines XML-Dokuments, die auf der abstrakten, logischen Struktur eines XML-Dokuments operiert. XPath-Ausdrücke verwenden eine Pfad-Notation, die es erlaubt durch die Knoten der hierarchischen Baumstruktur eines XML-Dokuments zu navigieren und Knoten mit bestimmten Eigenschaften und Werten aufzufinden und auszuwählen. Dabei kann unter anderem auf Elementknoten, Attributknoten oder Textknoten zugegriffen werden.

Die XPath-Syntax wird in XSLT-Dokumenten für die Auswahl von Knotenmengen durch den Vergleich mit einem bestimmten Muster (z.B. alle Elemente mit dem Attribut 'vowel') verwendet. Durch geeignete Formulierung der XPath-Ausdrücke in den XSLT-Anweisungen „select", „match" und „test" und die Kombination mit Operatoren (and, or, !, etc.) können die gewünschten Elemente oder Werte gesucht und die Ergebnisse als Zeichenkette zurückgegeben werden. Neben der Identifikation der zu bearbeitenden Teile eines Eingabedokuments, ist XPath auch für numerische Kalkulationen und die Manipulation von Strings geeignet.

In XIPA werden XPath-Ausdrücke genutzt, um die Elemente einer bestimmten Gruppe (z.B. alle Suprasegmentalia) aufzufinden und innerhalb der Tabelle in welcher das XSL-Template aufgerufen wurde zu verteilen. In einem weiteren Schritt werden für jedes Element die für die Anzeige relevanten Informationen extrahiert und in die entsprechenden Tabellenzellen eingetragen (Unicode-Zeichen, Beschreibung des Zeichens und – sofern verfügbar – ein Beispiel). Durch ein weiteres Template werden die zeichenspezifischen Meta-Informationen generiert, die den Tabellenzellen in Form von Attribut-Wert-Paaren angehängt werden. Diese Informationen bilden die Grundlage für die Benutzerinteraktion und die interaktiven Elemente die in XIPA mithilfe von JavaScript implementiert sind.

1.5 Ausgabe (client-seitige XSL-Transformation und XHTML)

Die Ausgabe der transformierten XML-Datei erfolgt im XML-basierten Format XHTML. Dieses Format muss den hohen Ansprüchen an XML-Dateien genügen. Demnach muss das Dokument wohlgeformt sein, d.h. es muss die Regeln der XML-Syntax einhalten, damit ein Parser problemlos mit dem Code umgehen kann. Weiterhin muss das Dokument valide (gültig) sein, d.h. die Syntax der verwendeten Elemente muss den Regeln und Standards des XML-Formats entsprechend eingesetzt werden (dies betrifft unter anderem die richtige Verwendung von Klammern, Elementverschachtelungen, sowie die Benennung von Tags und Attributen etc.).

Die strikte Formulierung in der Syntax und Semantik von XML bietet einige Vorteile. So ist die Kompatibilität mit älteren Browsern größtenteils gegeben, da diese das XHTML meist als einfaches HTML verarbeiten können. Weiterhin besteht in XHTML die Möglichkeit, Elemente aus anderen XML-basierten Sprachen direkt einzubetten. In XIPA bildet diese Möglichkeit die Grundlage, um die SVG-Grafik des Vokaltrapezes innerhalb der Transformation zu erstellen und um die Grafik unkompliziert mit den gleichen interaktiven Elementen wie den Rest der Seite bereitzustellen. Eine gewisse Zukunftssicherheit besteht unter anderem durch die Maschinenlesbarkeit des XHTML-Formats, die es ermöglicht, Abweichungen

oder Besonderheiten neuer Spezifikationen durch zusätzliche Stylesheets abzubilden und die Dokumente relativ einfach in das neue Format zu konvertieren.

Da die meisten modernen Browser einen XSLT-Prozessor bereitstellen, wurde sich dafür entschieden, die Transformation der XML-Datei vollständig auf den Client (d.h. das ausführende Gerät) auszulagern. Die Vorteile dieser Verfahrensweise liegen einerseits darin die Serverlast zu minimieren, das Potenzial aktueller Hardware zu nutzen und die Leistungsfähigkeit moderner Browser voll auszuschöpfen. Die Auslagerung auf den Client ist anderseits die Grundlage für die Nutzbarkeit des Tools im Offline-Modus und sichert die Funktionalität der Transkriptionsumgebung bei schwankender Leistung und Übertragungsgeschwindigkeit des Webservers.

1.6 JAVASCRIPT

Die interaktiven Elemente sind mit JavaScript realisiert. Für die Wahl von JavaScript sind sowohl die hohe Verbreitung von JavaScript-Interpretern in aktuellen Webbrowsern, als auch eine größere „Wartungsfreundlichkeit“ der Skripte ausschlaggebend[1]. Die Skripte sind in drei Dateien unterteilt. Um eine vereinfachte Nutzung der JavaScript Event-Handler zu ermöglichen, ist zusätzlich die JavaScript-Bibliothek jQuery eingebunden.

Die Entwicklung der JavaScript-Dateien wird in diesem Handbuch nur gestreift, da abzusehen ist, dass die Anpassung der Skripte den größten Veränderungen unterliegen wird. Im Folgenden wird die elementare Funktionsweise und Einbettung der Skripte unter der Angabe von Referenzen kurz erläutert.

1.6.1 JavaScript-Skripte

tooltip.js

Das JavaScript-Tooltip zur Anzeige der zusätzlichen Zeicheninformationen beim „Berühren“ der Zeichen mit der Maus basiert auf einer lizenzfreien Vorlage von Michael Leigeber. Die grundlegende Funktionsweise ist im Tutorial „Create a Nice, Lightweight JavaScript Tooltip“ erläutert.

Um die Funktionalität des Skripts an XIPA anzupassen, wurde der Code an einigen Stellen modifiziert. Die Veränderungen betreffen unter anderem:

- Größenanpassungen, um alle Inhalte innerhalb des Fensters darstellen zu können und um eine Darstellung über den Seitenrand hinaus zu verhindern.
- Positionsänderungen, um die Informationen rechts unterhalb des Mauszeigers zu positionieren und dabei möglichst wenig Inhalte durch die Einblendung des Tooltips zu verdecken. Da die Koordinaten im Skript relativ zum HTML-Element berechnet werden und XIPA im Browser zentriert dargestellt wird, muss der Abstand zum linken Seitenrahmen variabel berechnet werden. Um die vertikale Position des Mauszei-

1 Theoretisch wäre eine vollständige Implementierung der interaktiven Elemente in XQuery denkbar (W3C-Empfehlung einer Abfragesprache für XML-Daten). Auf diese Lösung wurde aufgrund der mangelnden (nativen) Browserunterstützung und der geringen Verbreitung von XQuery (und damit auch einer beschränkten Auswahl an Informationsquellen) verzichtet.

gers dynamisch zu erfassen und dabei Anpassungen von Fenstergröße und -breite zu berücksichtigen, wird der Abstand zwischen dem linken Fensterrand und dem Body-Element mit Hilfe der HTML-Eigenschaft `offset` dynamisch berechnet.

- Die Anzeige des Tooltips ist im CSS-Code angepasst und erfolgt ohne den Rückgriff auf Hintergrundgrafiken.

Die im Tooltip anzuzeigenden Informationen (z.B. Symbolname, Kardinalzahl, Unicode-Zeichen etc.) sind in der Funktion `toggleTooltip` im Skript `onclickSwitcher.js` definiert.

onclickSwitcher.js

Das Skript `onclickSwitcher.js` ist in erster Linie für den Wechsel zwischen Transkriptions- und Audioausgabemodus zuständig. Die für den Wechsel zwischen den Modi zuständigen Funktionen `switchToAudio` und `switchToTranscription` legen das Verhalten der Onclick-Handler für den jeweiligen Modus fest. Die Funktion `toggleTooltip` bestimmt, welche Informationen bei der Einblendung des Tooltips angezeigt werden sollen.

Das Skript bedient sich an jenen Informationen, die in den Metadaten der „klickbaren" Elemente hinterlegt sind. Diese bestehen aus einer Reihe von Attribut-Wert-Paaren (z.B. Symbolname, Unicode-Kode, Zeichenkodes für die unterschiedlichen Transkriptionsmethoden, dem Pfad zur Audiodatei etc.). Der Zugriff auf die Elemente (bzw. die Ausgabe der Attributwerte) erfolgt dabei über die verkürzte Schreibweise der jQuery-Methode `.attr(Attributname)`.

Im Abspielmodus (`switchToAudio`) wird die entsprechende Audiodatei durch Auslesen der Pfadangabe im Attribut `data-sound` geladen und mittels `XMLHttpRequest` angefragt. Sofern die entsprechende Audiodatei im Verzeichnis vorhanden ist, wird diese in den zu Beginn der Skript-Datei definierten Audio-Kontext eingehängt und abgespielt (siehe Abschnitt 1.6.2 Web-Audio-API).

Beim Wechsel in den Transkriptionsmodus (`switchToTranscription`) wird der Wert der im Dropdown-Menü eingestellten Ausgabemethode (Unicode, TeX etc.) ausgelesen. In Abhängigkeit von der gewählten Transkriptionsmethode werden die entsprechenden Attributwerte – also die Kodes für Unicode, Unicode-Entities, LaTeX- oder Praat-Notation – mit dem Onklick-Handler verknüpft. Sobald ein Element geklickt wird, wird der Wert zur weiteren Verarbeitung an die Funktion `insertAtCursor` im gleichnamigen Skript weitergegeben.

insertAtCursor.js

Im Skript `insertAtCursor.js` ist die Funktion `insertAtCursor` definiert. Diese ist für das Einfügen des Zeichens (bzw. die Ausgabe des Kodes) an der aktuellen Cursorposition im Textfeld zuständig. Das Skript bietet die Möglichkeit, Zeichen und Diakritika per Mausklick in das Textfeld zu übertragen. Bei aktivem (schwarz hinterlegtem) Textfeld kann die Cursorposition zwischen den Eingaben mit den Pfeiltasten oder der Maus verändert werden. Die zuletzt gewählte Position des Cursors bleibt auch erhalten, wenn das Textfeld durch einen Mausklick auf einen der inaktiven Bereiche der Seite oder einen Wechsel zwischen Transkriptions- und Abspielmodus im passiven Zustand ist.

Das Skript basiert auf einer Vorlage von Torsten Anacker. Die Grundzüge der Funktionsweise des Skripts sind im Artikel „Formulare: Text an Cursorposition einfügen" erläutert. An einigen Stellen ist das Skript gekürzt und an die gegebenen Anforderungen angepasst.

1.6.2 Web Audio API

Die Web Audio API ist eine Programmierschnittstelle zur Verarbeitung, Wiedergabe und Manipulation von Audiodateien in Web-Anwendungen, die das direkte Laden und Weiterverarbeiten von Audiodateien per JavaScript in Web-Anwendungen ermöglicht. Die Nutzung der Web Audio API ist ohne die Installation zusätzlicher Plug-ins möglich und bietet einige Funktionen, die über die grundlegenden Abspielfunktionen des HTML5 `audio`-Elements hinausgehen. Neben der reinen Wiedergabe- bzw. Abspielfunktion werden unter anderem die Visualisierung von Audio-Rohdaten, die Synthese von Klängen (durch Oszillatoren) und die Manipulation der Wiedergabe (durch diverse Filter) unterstützt. Im folgenden Verlauf sind die für XIPA relevanten Eigenschaften der Wiedergabefunktion der Web Audio API kurz skizziert.

Um die auf dem Server oder der Festplatte gespeicherten Audiodateien abspielen zu können, muss in erster Instanz ein `AudioContext` bereitgestellt werden. In diesen Kontext können mehrere Objekte eingehängt werden, die in Form von `AudioNode`-Objekten repräsentiert werden. Der Aufruf des Audio Kontexts und die Wiedergabefunktion (`switchToAudio`) sind im Skript `onclickSwitcher` abgelegt.

Die Quelle (d.h. die beim Mausklick auf einen der Laute abzuspielende Datei), wird im Skript mittels `XMLHttpRequest` geladen. Dazu wird der im `data-sound`-Attribut hinterlegte Pfad über die Variable `filename` ermittelt. Wenn die Quelldatei existiert, wird dem `AudioContext` die Methode `createBufferSource()` zugewiesen – sollte die Datei nicht gefunden werden, wird die Anfrage mit einer Fehlermeldung quittiert. Beim Laden der Audiodatei wird die Methode `decodeAudioData()` des `AudioContext` aufgerufen, um die betreffende Audiodatei zu dekodieren. Im weiteren Verlauf wird die Audiodatei in den Puffer geladen und abgespielt.

Obwohl die Web Audio API von den meisten aktuellen Browsern unterstützt wird, bestehen hinsichtlich der Umsetzung in den verschiedenen Browsern einige Unterschiede. Es ist zu erwarten, dass diese sich im Laufe der Entwicklung der W3C Empfehlung an einen Quasi-Standard angleichen werden.

Das System besteht aus drei grundlegenden Dateien: einer XML-Datei (`IPA.xml`), welche die Informationen zu allen hinterlegten Zeichen enthält, einer Dokumenttyp-Definition (`ipa.dtd`), durch welche die erlaubten Einträge in der XML-Datei definiert werden und einem XSL-Stylesheet (`IPA.xsl`), das für die Transformation der XML-Daten in ein vom Browser lesbares Format – in diesem Fall (X)HTML – verantwortlich ist.

Diese Dateien werden durch drei JavaScript Skripte ergänzt (`insertAtCursor.js`, `onclickSwitcher.js` und `tooltip.js`). Die Skripte sind für die Funktionalität, d.h. das Einfügen von Text an der Cursorposition, Verknüpfung und Wechsel zwischen Event-Handlern (Anwenderereignissen) für die Wiedergabe von Audio und das Einfügen des Zeichen-Codes bei Mausklick und die Anzeige der erweiterten Informationen beim Bewegen des Mauszeigers über die Symbole verantwortlich.

Um die Logik des XSL-Stylesheets vom Layout zu trennen, wurden (wenn möglich) die meisten Style-Elemente in eine weitere XSL-Datei (`attributes.xsl`) und ein CSS-Stylesheet (`styles.css`) ausgegliedert.

Alle Dateien wurden manuell erstellt und mit unterschiedlichen XML-Editoren und (validierenden) XML-Parsern auf Wohlgeformtheit und Validität überprüft. Funktionalität und Erscheinungsbild wurden über die internen XSLT-Prozessoren von Mozilla Firefox, Google Chrome und Internet-Explorer Edge auf unterschiedlichen Betriebssystemen getestet und angepasst.

Ordnerstruktur im Projekt

```
xipa            <!-- Hauptordner -->
-- .htaccess    <!-- Server-Konfigurationsdatei (Medientypen-Anweisungen, Browser-Caching-Richtlinien) -->
-- IPA.xml      <!-- Hauptdatei des Projekts (im Web-Browser ausführbar) -->
   > data                       <!-- 1. Unterordner -->
   -- IPA.xsl                   <!-- XSL-Stylesheet (Layout-Vorgaben und Templates)                    -->
   -- IPA.dtd                   <!-- Dokumenttyp-Definition (legt die erlaubten Elemente fest)         -->
   ... attributes.xsl, vowels.xsl, styles.css   <!-- weitere Dateien -->
     >> audio_samples
          >>> mp3                 <!-- *.mp3 Dateien der Laute (Reduktion von Serverlast und Traffic)  -->
          >>> wav                 <!-- unkomprimierte Audiodateien *.wav Format                        -->
          ... 00E6.wav, 00F8.wav ...
     >> fonts
        ... CharisSIL-R.woff                      <!-- Webfont-Version der Charis SIL Schriftart      -->
     >> scripts                                   <!-- 3 JavaScript-Skripte und die jQuery Bibliothek -->
        ... insertAtCursor.js, tooltip.js, onclickSwitcher.js, jquery-1.11.3.min.js
```

2.1 XML–DATEN (IPA.XML)

Datengrundlage des Systems bildet die Datei `IPA.xml`. In ihr sind sämtliche Zeichen und Informationen hinterlegt. Zusätzlich wurden für eventuell hinzukommende Zeichen (insbesondere bei Konsonanten und Vokalen) Leerstellen gelassen, die einen Nachtrag an der jeweiligen Position erleichtern.

Im Header der XML-Datei befindet sich die Angabe der zugrunde liegenden XML-Version (XML 1.0 Spezifikation), der verwendeten Zeichenkodierung (UTF-8-Kodierung), der URI der Dokumenttyp-Deklaration (d.h. der Bezug zur externen Dokumenttyp-Definition, in welcher Angaben zu Struktur und Regeln der XML-Datei definiert sind) sowie der Verweis auf das zur Weiterverarbeitung (zur Transformation) zu verwendende Stylesheet.

Aufbau der XML-Datei (IPA.xml)

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

<!DOCTYPE ipa SYSTEM "data/ipa.dtd">                    <!-- Dokumenttyp-Deklaration  -->
<?xml-stylesheet type="text/xsl" href="data/IPA.xsl"?>  <!-- Link zum XSL-Stylesheet  -->

<ipa>                                                   <!-- Wurzelelement -->
  <sound>...</sound>                                    <!-- Definition der einzelnen Laut-Elemente    -->
  <sound>...</sound>                                    <!-- Definition der einzelnen Laut-Elemente... -->
 ...
</ipa>
```

Auf das Wurzelelement `<ipa>` folgen aneinandergereiht alle Laute des Systems. Jeder Laut ist in ein `<sound>`-Element eingebettet, das den Vorgaben der DTD entspricht und ist mit einem Attribut `type` versehen, das die jeweilige Kategorie (Konsonant, Vokal, Diakritikum etc.) festlegt.

Beispiel für ein sound–Element eines Konsonanten aus der Datei IPA.xml

```
<sound type="consonant">
 <manner>plosive</manner>          <!-- Artikulationsart            -->
 <place>glottal</place>            <!-- Artikulationsort            -->
 <category>voiceless</category>    <!-- stimmhaft / stimmlos        -->
 <class>pos</class>                <!-- possible,empty,impossible   -->
   <name>
    <symbol>Glottal stop</symbol>  <!-- Name des Symbols            -->
    <tex>\\textglotstop</tex>      <!-- TeX-Befehl                  -->
    <praat>\\?g</praat>            <!-- Praat-Code                  -->
    <ucentity>0294</ucentity>      <!-- Unicode Zeichen (UCS Code)  -->
    <unicode>&#x294;</unicode>     <!-- Unicode-Zeichen             -->
    <cardinal>113</cardinal>       <!-- Kardinalzahl (IPA Number)   -->
   </name>
</sound>
```

2.2 DOKUMENTTYP–DEFINITION (IPA.DTD)

Die Einträge innerhalb der einzelnen `<sound>`-Elemente der XML-Datei (`IPA.xml`) entsprechen den Vorgaben der externen Dokumenttyp-Definition (`ipa.dtd`) im Unterordner `/data`, mit welcher das XML-Dokument verknüpft ist. Die DTD legt das XML-Dokument als eine Datei fest, die aus mehreren Elementen des Typs `<sound>` besteht, welche wiederum die Elemente `type`, `manner`, `place` etc. (Typ, Art, Ort etc.) enthalten können. Ein `sound`-Element muss mit einem Attribut `type` versehen sein. Zusätzlich müssen die Elemente `manner` und `name` angegeben werden. Alle weiteren Angaben sind optional. Die Attribute der `sound`-Elemente müssen mit einem der angegebenen Typen aus der Attributliste (Konsonant, Vokal, Diakritikum etc.) übereinstimmen.

Exemplarischer Aufbau der Dokumenttyp-Definition (ipa.dtd)

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ipa (sound+)>
```

```
<!ELEMENT sound (type, manner, place?, category?, class?, name, name2?, example1?, example2?, ucentity?,
    symbol?, praat?)>
<!ATTLIST sound
          type (consonant | vowel | diacritic | other | suprasegmental | consonantnp | toneoraccent) #
    REQUIRED>

<!ELEMENT type     (#PCDATA)>
<!ELEMENT manner   (#PCDATA)>
<!ELEMENT place    (#PCDATA)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT class    (#PCDATA)>
<!ELEMENT name (symbol?, tex, tex2?, praat?, ucentity?, unicode, cardinal, sampa?)>
...
```

Innerhalb des `<sound>`-Elements müssen die grundlegenden Informationen zu Artikulationsart, -ort und -modus hinterlegt sein. Die Angabe der Klasse (`class`) definiert, ob es sich bei dem Eintrag um einen existierenden Laut (`pos`=possible/möglich), um einen nicht vorhandenen aber theoretisch möglichen Laut (`emp`=empty/leer), oder um einen als nicht artikulierbar klassifizierten Laut (`imp`=impossible/unrealisierbar) handelt.

Existierende Laute (bzw. Symbole) haben mindestens ein weiteres Element mit der Bezeichnung `name`. In diesem Element sind die zugehörigen Zeicheninformationen hinterlegt, die Informationen zum Unicode-Zeichen in hexadezimaler Schreibweise (z.B. ʔ), zur Kardinalzahl des Symbols im Alphabet der IPA (z.B. 113) und einer Angabe zur Notation in LaTeX (z.B. \textglotstop) bestehen müssen. Alle weiteren Angaben – wie beispielsweise die Notation in Praat oder SAMPA – sind optional gehalten. Dies ist notwendig, da diese Notationsarten nicht für jedes Symbol des IPA eine Entsprechung bereithalten.

2.3 XSL-Stylesheet (IPA.xsl)

Das XSL-Stylesheet bestimmt maßgeblich die Ausgabe des Dokuments. Im Header des Stylesheets wird die verwendete XML-Version und UTF-8 Zeichenkodierung festgelegt. Der Eintrag `<xsl:stylesheet...>` deklariert, dass es sich um ein Stylesheet der XSLT 1.0 Spezifikation handelt, welches Anweisungen aus dem Namensraum der XSL-Transformationen beinhaltet. Als Ausgabemethode ist HTML festgelegt.

Struktur der XSL-Datei (IPA.xsl)

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsl:stylesheet version="1.0"                          <!-- Anfang des XSL-Stylesheets (XSLT Vers. 1.0) -->
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform" <!-- Angaben zu verwendeten Namensraum URIs      -->
      xmlns:svg="http://www.w3.org/2000/svg"
      xmlns:xhtml="http://www.w3.org/1999/xhtml"
      xmlns:xlink="http://www.w3.org/1999/xlink">

      <xsl:output method="html" encoding="UTF-8"/>    <!-- Definition der Ausgabemethode und Kodierung  -->
      <xsl:include href="./attributes.xsl" />         <!-- Einbinden externer Dateien (XSL-Stylesheet)  -->

        <xsl:template match="/">                      <!-- "match-all"-Template -->
          <html>
            <head>...</head>                          <!-- Angabe der Kopfdaten (s. nächster Abschnitt) -->
            <body>
              ...     <!-- Anweisungen zur Formatierung der Grundstruktur und Aufrufe weiterer Templates -->
                 <xsl:apply-templates select="sound"/>
             </body>
          </html>
        </xsl:template>                               <!-- Ende des "match-all"-Templates -->

                                                      <!-- Anfang des Template-Teils (siehe Templates)  -->
```

```
      <xsl:template name="xyz">
      ...                    <!-- beinhaltet diverse Anweisungen, kann selbst weitere Templates aufrufen -->
      </xsl:template>
      ...
 </xsl:stylesheet>                                    <!-- Ende des XSL-Stylesheets -->
```

2.3.1 Head- und Body-Tags im Stylesheet

Die XSL-Datei besteht aus zwei Teilen. Im ersten Teil des Stylesheets wird der Grundaufbau der Seite definiert. Hier befinden sich innerhalb der `html`-Tags die Elemente `head` und `body`.

In den Kopfdaten (dem `head`-Element) werden zunächst Metadaten des Dokuments (z.B. ein Titel) angegeben. Zusätzlich können hier weitere Meta-Informationen über das Dokument (z.B. Schlüsselwörter und Beschreibungen, verwendete Zeichenkodierung etc.) hinterlegt werden. Das `head`-Element beinhaltet weiterhin alle für den Browser relevanten Instruktionen und Informationen, z.B. Styling-Angaben, die durch Einbinden eines externen CSS-Stylesheets referenziert werden und die Aufrufe von Skript-Dateien, die für die Funktionalität des Dokuments beim Seitenaufruf bereitstehen müssen.

Auszug aus dem head-Element der XSL-Datei (IPA.xsl)

```
<head>
 <title>[XIPA] - Extensible International Phonetic Alphabet</title>     <!-- Definition des Titels      -->
 <meta name="keywords" content="transcription, phonetics, IPA, ..." />

 <link rel="stylesheet" type="text/css" href="./data/styles.css" />     <!-- Pfad zum CSS-Stylesheet    -->
 <script type="text/javascript" src="./data/tooltip.js"></script>      <!-- Einbinden externer Skripte -->
 <script>...</script>
 ...
</head>
```

Innerhalb der `<body>`-Tags werden einzelne Elemente, z.B. (vorerst leere) Tabellen, das Grundgerüst der Vektorgrafik für das Vokaltrapez sowie das Eingabefeld erstellt und auf der Seite angeordnet. Mit dem Befehl `xsl:call-template...`, welcher auf den Template-Teil des Dokuments referenziert, können die Elemente mit ihrem jeweils zugehörigen Inhalt gefüllt werden. Attribute (z.B. IDs oder „Style“-Angaben etc.) können innerhalb des XSL-Dokuments selbst gesetzt werden, in das externe CSS-Stylesheet ausgelagert werden, oder aber aus der im Header der XSL-Datei eingebundenen Stylesheet-Datei (`attributes.xsl`) in Form von vordefinierten „Attribut-Sets“ abgerufen werden.

Auszug aus dem body-Tag der XSL-Datei (IPA.xsl)

```
<body xsl:use-attribute-sets="body.style">  <!-- xsl:use-attribute-sets importiert Style-Anweisungen    -->
                                            <!-- eingebundene Attribut-Sets z.B.: "body.style" ...      -->
                                            <!-- die Sets sind in der Datei attributes.xsl definiert    -->
 <script>...</script>  <!-- inline JavaScript Code (z.B. Testen der Web-Audio-API-Kompatibilität etc.) -->

<!-- Definitionen aller darzustellenden Elemente (Tabellen, SVG, Textfeld etc.) des Ergebnis-Dokuments -->
...
  <div xsl:use-attribute-sets="div.suprasegmentals" id="suprasegmentals">
     <caption xsl:use-attribute-sets="caption.style"><h3>Suprasegmentals</h3></caption>
       <table id="sup" xsl:use-attribute-sets="table.suprasegmentals">
          <!-- Aufruf des Templates mit dem Namen "Suprasegmentals" innerhalb des Tabellen-Elements    -->
          <xsl:call-template name="Suprasegmentals"/>
       </table>
  </div>
...
</body>
```

2.3.2 Templates

Der zweite Teil der XSL-Datei – d.h. der Abschnitt nach dem „match-all“-Template, in welchem der HTML-Teil der Ergebnisdatei definiert ist – besteht aus einer freien Abfolge von Template-Regeln. Nach der letzten Anweisung im `body`-Tag (`xsl:apply-templates select=...`) werden alle Knoten des Typs `sound` zur weitergehenden Bearbeitung ausgewählt. Ein vereinfachtes Beispiel für die Verarbeitung eines Template-Aufrufs (`xsl:call-template...`) am Beispiel der Suprasegmentalia findet sich im folgenden Ausschnitt:

Beispiel-Template aus dem Anweisungsteil der XSL-Datei (IPA.xsl)

```
<!-- ########################  TEMPLATES  ######################### -->
...
<xsl:template name="Suprasegmentals">
 <xsl:for-each select="//sound[@type='suprasegmental']">
 <!-- xsl:for-each führt die folgenden Anweisungen für alle gewählten Elemente aus   -->
 <!-- Ausgewählt werden alle <sound>-Elemente mit dem Attribut type='suprasegmental' -->
  <tr>
    <td class="{class/text()}" width="18%" xsl:use-attribute-sets="no.border.right">
        <!-- class=... ruft die Klasse des jeweiligen Elements auf, sofern vorhanden ist diese "pos"  -->
        <!-- das Attribut-Set "no.border.right" erstellt das Attribut style="border-right: 0em none;" -->
        <xsl:call-template name="Transcribe">   <!-- ruft das Template mit dem Namen "Transcribe" auf -->
           <xsl:with-param name="name" select="./name"></xsl:with-param>
        </xsl:call-template>
    </td>
  </tr>
 </xsl:for-each>
</xsl:template>
...
  <!-- Das Template "Transcribe" füllt die Zelle mit dem entsprechenden Inhalt (s. nächster Abschnitt) -->

</xsl:stylesheet>
```

Der Aufruf `xsl:call-template name=...` aus dem `body`-Element ruft das Template mit dem Namen `Suprasegmentals` aus dem Template-Abschnitt der XSL-Datei auf. Die `xsl:for-each`-Anweisung filtert mittels XPath-Ausdruck (`select="//sound[@...]"`) diejenigen `sound`-Elemente aus der XML-Datei, die das Attribut `type='suprasegmental'` aufweisen. In die – schon vor Template-Aufruf definierte – Tabelle wird nun für jedes gefundene Element eine neue Tabellenzeile (`tr`) erstellt und die Zelle (`td`) mit dem entsprechenden Zelleninhalt (in diesem Fall dem Unicode-Zeichen) versehen.

Die eigentliche Vorgehensweise ist komplexer als hier dargestellt, da jede Zeile meist noch mit zusätzlichen Informationen wie der Bezeichnung des Symbols und ggf. Beispielen angereichert wird. Eine Darstellung der einzelnen Elemente und Bausteine ist schematisch in Abschnitt 3 dargestellt.

2.4 TEMPLATES ZUR AUSGABE DER UNICODE-ZEICHEN

Für die Ausgabe und Zuordnung der Unicode-Zeichen sind zwei Templates zuständig – die Templates „`Transcribe`“ und „`Plot_Symbol`“. Beide befinden sich im oberen Teil des Template-Abschnitts des XSL-Stylesheets.

2.4.1 Das Template Transcribe (Tabellen)

Innerhalb aller Templates (Suprasegmentals, Diacritics, Accents etc.) wird das Template `Transcribe` aufgerufen sobald in einer Zelle das Unicode-Zeichen ausgegeben werden soll. Die Unicode-Kodes (bzw. die Unicode-Entities) der Grundzeichen sind in der XML-Datei innerhalb des `name`-Knotens im `unicode`-Tag hinterlegt. Da für Beispiele – diese liegen im `example1`- oder `example2`-Knoten – ebenfalls Angaben des Unicode-Kodes existieren, wird zur Bestimmung des Pfades ein Parameter übergeben, der das jeweils zutreffende Zeichen lokalisiert. Dieser Pfad wird bereits beim Aufruf des Templates `Transcribe` übergeben. Soll eine Tabellenzelle mit dem Inhalt des ersten Beispiels gefüllt werden, wird statt des Parameters `./name` der Parameter `./example1` übergeben.

Das Template Transcribe

```
<!-- ########################## PLOT UNICODE SYMBOLS ########################## -->
<xsl:template name="Transcribe">
  <xsl:param name="name"/>

    <xsl:call-template name="Add_metadata">
      <xsl:with-param name="name" select="$name" />      <!-- leitet den Pfad des name-Attributs weiter -->
    </xsl:call-template>

       <xsl:value-of select="($name/unicode"/>
       <!-- Ausgabe der im unicode-Tag hinterlegten Unicode-Zeichen in allen Tabellenzellen            -->
</xsl:template>
```

Der Ausgabewert wird durch die `select`-Anweisung initiiert. Dabei legt `$name` den Pfad fest, aus welchem das Zeichen gewählt werden soll. Die Angabe von `/unicode` navigiert letztlich zum `unicode`-Element, dessen Inhalt daraufhin in die Ausgabe geschrieben wird.

2.4.2 Das Template Plot Symbol (SVG-Grafik)

Da die Vektorgrafik etwas anders strukturiert ist, konnte hier das `Transcribe`-Template nicht vollständig wiederverwertet werden. Das Template erfüllt zwar auch hier seine Aufgabe, die Elemente mit Metadaten anzureichern (durch den Aufruf des Templates `Add_metadata`), die eigentliche Ausgabe der Zeichen innerhalb der Grafik ist jedoch in ein weiteres, einfaches Template – das Template `Plot_Symbol` – ausgelagert.

Die Gründe für die separate Ausgabe der Zeichen liegen darin, dass die Vektorgrafik aus mehreren Ebenen aufgebaut ist und eine geringfügig abgeänderte Syntax verlangt. Um zu verhindern, dass der Nutzer das Zeichen exakt treffen muss, wird unterhalb jedes Symbols ein Kreis-Element generiert, um die „klickbare“ Fläche zu vergrößern. Mit dieser Fläche sind die Metainformationen verknüpft. Das sichtbare Zeichen liegt eine Ebene höher. Der Aufbau der Vektorgrafik ist in Abschnitt 3.3 skizziert.

Das Template Plot_Symbol

```
<!-- ######################## PLOT VOWEL SYMBOLS IN SVG ######################## -->
<xsl:template name="Plot_Symbol">
  <xsl:param name="name"/>
    <xsl:value-of select="$name/unicode/text()"/>
    <!-- Ausgabe des Unicode-Zeichens innerhalb der SVG-Grafik -->
</xsl:template>
```

2.5 DAS METADATEN-TEMPLATE

Zusätzliche Attribute für die Weiterverarbeitung mit Skripten werden durch das Template `Add_metadata` erstellt. Dieses Template weist jedem `sound`-Element mit der Klasse `class='pos'` – d.h. jedem nicht-leeren Element – eine Reihe an Attributen zu, die für die Funktionalität entscheidend sind. Das Metadaten-Template befindet sich am Ende der Datei `IPA.xsl`. In Abschnitt 3 ist das Zusammenwirken der Templates (vom Aufruf innerhalb des `body`-Tags bis hin zu den Aufrufen der Templates für die Ausgabe der Unicode-Zeichen und der Metadaten) für einige Elemente der IPA-Tabelle schematisch skizziert.

Das Metadaten-Template (Add_metadata)

```
<!-- ########################## ADD METADATA TEMPLATE ######################### -->
<xsl:template name="Add_metadata">
  <xsl:param name="name"/>

<xsl:if test="class = 'pos'">              <!-- Auswahl der Elemente der Klasse 'pos'                  -->
                                           <!-- (alle Elemente mit hinterlegtem Unicode-Zeichen)       -->
  <xsl:attribute name="symbol_name">       <!-- erstellt ein Attribut mit dem Namen symbol_name        -->
    <xsl:value-of select="$name/symbol"/>  <!-- Attributwert ist der im <symbol>-Tag hinterlegte Text  -->
  </xsl:attribute>                         <!-- (z.B. "Glottal stop") -->

  <xsl:if test="$name/cardinal != ''">     <!-- wenn der Inhalt des <cardinal>-Tags nicht leer ist...  -->
    <xsl:attribute name="ipa_number">      <!-- erstelle ein weiteres Attribut mit dem Namen ipa_number -->
      <xsl:value-of select="$name/cardinal"/>
    </xsl:attribute>
  </xsl:if>

  <xsl:attribute name="unicode">           <!-- erstellt ein weiteres Attribut mit dem Namen unicode   -->
    <xsl:value-of select="$name/unicode"/> <!-- Ausgabewert ist der aufgelöste Wert der Entity-Referenz -->
  </xsl:attribute>                         <!-- ... wird bei der Ausgabemethode "Unicode" ausgegeben   -->

  <xsl:attribute name="praat">
    <xsl:choose>  <!-- Teste, ob der Eintrag der Praat-Notation "Escape-Sequenzen"  (z.B. \) beinhaltet -->
      <xsl:when test="$name/praat[starts-with(text(), '\')]">     <!-- in IPA.xml z.B. als \\xx notiert -->
        <xsl:value-of select="substring($name/praat/text(), 2)"/> <!-- wenn ja, überspringe ersten \    -->
      </xsl:when>
      <xsl:otherwise>                                <!-- andernfalls: Ausgabe des vollständigen Kodes -->
        <xsl:value-of select="$name/praat/text()"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>

 <xsl:attribute>...</xsl:attribute>  <!-- weitere Attribute ... -->
                                     <!-- z.B. "data-sound"-Attribut (Angabe des Pfades zur Audiodatei) -->
</xsl:if>

</xsl:template>
```

Metadaten-Attribute der resultierenden (X)HTML-Datei am Beispiel des Glottal Stop

```
<td class="pos"                                  <!-- öffnendes <td>-Element der Klasse "pos" -->
    style="border-right: 0em none;"                 <!-- importierte Style-Regeln für das Element -->
    symbol_name="Glottal stop"                      <!-- Name des Symbols -->
    phonetic_description="Voiceless Glottal Plosive" <!-- Phonetische Beschreibung des Lauts -->
    unicode_entity="0294"                           <!-- Unicode Zeichenreferenz -->
    ipa_number="113"                                <!-- Kardinalzahl (IPA-Nr.) -->
    data-sound="./data/audio_samples/wav/0294.wav"  <!-- Pfad zur verknüpften Audiodatei -->
    unicode="&#x294;"                               <!-- Unicode-Zeichen (dargestellt wird das
     umgewandelte Symbol)-->
    tex="\textglotstop"                             <!-- ausführliche LaTeX-Notation -->
    tex2="P"                                        <!-- kurze LaTeX-Notation -->
    praat="\?g">                                    <!-- Praat-Notation -->
    &#x294;                                         <!-- sichtbares Unicode-Zeichen im <td>-Element (
     Symbol)-->
</td>                                            <!-- schließendes <td>-Element -->
```

Die meisten Elemente der IPA-Tabelle sind in Tabellenform angelegt. Aus diesem Grund werden in diesem Abschnitt lediglich einige stereotype Konzepte zum Aufbau erläutert, die sich in ähnlicher Form in anderen Templates wiederfinden. Da sich der Aufbau der Konsonantentabelle und der Aufbau des Vokaltrapezes zum Teil anderer Methoden bedienen, sind sie in diesem Kapitel separat aufgeführt.

3.1 KONSONANTENTABELLE („CONSONANTS“)

Auszüge aus der Konsonanten-Tabelle aus dem body-Tag und dem „Consonants“-Template

```
<div xsl:use-attribute-sets="div.consonants" id="consonants">
<caption xsl:use-attribute-sets="caption.style"><h3>Consonants (Pulmonic)</h3></caption>
  <table id="con" xsl:use-attribute-sets="table.consonants">
    <tr xsl:use-attribute-sets="space.adjustment">         <!-- Überschriftenzeile mit angepasster Höhe -->
      <th xsl:use-attribute-sets="th.manner_and_place"></th>  <!-- Tabellenspalte für Artikulationsart -->
      <th xsl:use-attribute-sets="th.place">Bilabial</th>     <!-- neue Spalte für bilabiale Laute     -->
      <th xsl:use-attribute-sets="th.place.large">Labiodental</th><!-- breitere Spalte (längerer Text) -->
      ...
    <tr manner="plosive">                                    <!-- neue Tabellenzeile für die Plosive   -->
      <th xsl:use-attribute-sets="th.manner">Plosive</th>    <!-- setzt die Zeilenüberschrift: Plosive -->
        <xsl:call-template name="Consonants">                <!-- 1. Aufruf des "Consonants"-Templates -->
          <xsl:with-param name="manner" select="'plosive'"/> <!-- übergebener Parameter 'plosive'      -->
        </xsl:call-template>          <!-- alle gefundenen Elemente werden der Reihe nach abgearbeitet -->
    </tr>
    <tr manner="nasal">                                      <!-- neue Tabellenzeile für nasale Laute  -->
      <th xsl:use-attribute-sets="th.manner">Nasal</th>
        <xsl:call-template name="Consonants">                <!-- 2. Aufruf des "Consonants"-Templates -->
          <xsl:with-param name="manner" select="'nasal'"/>   <!-- übergebener Parameter 'nasal'        -->
        </xsl:call-template>
    </tr>
    ...
  </table>
</div>

<!-- ########################  TEMPLATE CONSONANTS  ######################### -->
  <xsl:template name="Consonants">
    <xsl:param name="manner"/>      <!-- liest den übergebenen Wert des Parameters aus (z.B. 'plosive') -->
    <xsl:for-each select="ipa/sound[@type='consonant' and manner=$manner]">      <!-- for-each-Schleife -->
      <xsl:choose> <!-- für jedes gefundene Element vom Typ 'consonant' und dem jeweiligen Parameter... -->
        <xsl:when test="category='voiceless'"> <!-- test: wenn die Kategorie des Lautes 'voiceless' ist -->
          <td class="{class/text()}" xsl:use-attribute-sets="no.border.right"><!-- kein Rahmen rechts -->
            <xsl:if test="name/unicode != ''">     <!-- wenn das name/unicode-Element nicht leer ist... -->
            <xsl:call-template name="Transcribe">  <!-- Aufruf des Transcribe Templates (vgl. 3.4, 3.5) -->
              <xsl:with-param name="name" select="./name"></xsl:with-param>    <!-- Übergabe des Pfades -->
            </xsl:call-template>         <!-- (Aufruf von Transcribe führt zum Aufruf von Add_metadata) -->
            </xsl:if>  <!-- Ausgabe des Unicode-Zeichens durch <xsl:value-of select="./name/unicode"/>  -->
          </td>        <!-- Ende der Tabellenzelle - wenn kein Zeichen existiert, bleibt die Zelle leer -->
        </xsl:when>
        <xsl:when test="category='voiced'">  <!-- gleiches Vorgehen mit dem Attribut-Set no.border.left -->
        ...
        </xsl:when>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
```

3.2 Weitere Tabellen (am Bsp. der Tabelle „OtherSymbols“)

Das Template `OtherSymbols` dient prototypisch der Beschreibung aller Elemente, die zusätzlich eine Ausgabe der Lautbezeichnung, der Kategorie des Lautes oder eine zusätzliche Beschreibung beinhalten.

Erstellung der Tabelle „Other Symbols“

```
<!-- ############################  TABLE OTHER SYMBOLS ########################## -->
<div xsl:use-attribute-sets="div.other_symbols" id="other_symbols">
<caption xsl:use-attribute-sets="caption.style"><h3>Other Symbols</h3></caption>     <!-- Überschrift -->
  <table id="oth" xsl:use-attribute-sets="table.other_symbols"><!-- Abruf der passenden Style-Attribute -->
     <xsl:call-template name="OtherSymbols"/>                  <!-- Aufruf des "OtherSymbols"-Templates -->
   </table>
</div>

<!-- ########################  TEMPLATE OTHER SYMBOLS  ######################### -->
<xsl:template name="OtherSymbols">

    <xsl:for-each select="//sound[@type='other']">         <!-- für jedes Element vom Typ 'other' wird: -->
     <tr>                                                  <!-- eine neue Zeile in der Tabelle erstellt -->
     <xsl:choose>
      <xsl:when test ="name2 != ''">          <!-- wenn ein <name2>-Element existiert, dann zwei Spalten-->
        <td class="{class/text()}" width="10%" xsl:use-attribute-sets="no.border.right"> <!-- 1. Spalte -->
           <xsl:call-template name="Transcribe">             <!-- in die erste (schmale) Zelle wird das -->
             <xsl:with-param name="name" select="./name"></xsl:with-param><!-- <name>-Element eingefügt -->
           </xsl:call-template>
       </td>
        <td class="{class/text()}" width="10%" xsl:use-attribute-sets="no.borders">      <!-- 2. Spalte -->
         <xsl:call-template name="Transcribe">              <!-- in die zweite (schmale) Zelle wird das -->
           <xsl:with-param name="name" select="./name2"></xsl:with-param><!-- <name2>-Element eingefügt -->
         </xsl:call-template>
        </td>
      </xsl:when>
      <xsl:otherwise>         <!-- wenn kein <name2>-Element existiert, wird eine breite Spalte erzeugt -->
        <td class="{class/text()}" width="20%" colspan="2" xsl:use-attribute-sets="single.symbol">
           <xsl:call-template name="Transcribe">        <!-- in die breite, zweispaltige Zelle wird das -->
             <xsl:with-param name="name" select="./name"></xsl:with-param><!-- <name>-Element eingefügt -->
           </xsl:call-template>
       </td>
      </xsl:otherwise>
     </xsl:choose>
  <td class="emp" width="80%" xsl:use-attribute-sets="td.text.style"> <!-- neue Spalte für Beschreibung -->
     <xsl:if test="category != ''">              <!-- Ausgabe der Kategorie (wenn diese nicht leer ist) -->
    <!-- Inhalt des Kategorie-Elements wird ausgegeben, 1. Zeichen wird in Großbuchstaben transformiert -->
        <xsl:value-of select="concat(substring(translate(./category/text(),$lowercase,$uppercase),1,1),
     substring(./category/text(),2))"/>
    <!-- Großschreibung des ersten Zeichens (durch Aufruf der Variablen 'lowercase' und 'uppercase')    -->
     </xsl:if>
     <xsl:if test="category = ''">               <!-- wenn das Kategorie-Element des Zeichens leer ist -->
       <xsl:text> </xsl:text>               <!-- Ausgabe eines Leerzeichens und des Artikulationsortes -->
          <xsl:value-of select="concat(substring(translate(./place/text(),$lowercase,$uppercase),1,1),
     substring(./place/text(),2))"/>
     </xsl:if>
     <xsl:if test="category != ''">  <!-- Anfügen weiterer Angaben (bei nicht-leerem Kategorie-Eintrag) -->
       <xsl:text> </xsl:text>                <!-- Leerzeichen (folgt hinter der Ausgabe der Kategorie) -->
         <xsl:value-of select="place/text()"/>                    <!-- Ausgabe des Artikulationsortes -->
     </xsl:if>
       <xsl:text> </xsl:text>
          <!-- Ausgabe des Artikulationsmodus (bzw. der im 'manner'-Element hinterlegten Beschreibung) -->
         <xsl:value-of select="manner/text()"/>
       </td>
    </tr>

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

3.3 VEKTORGRAFIK DER VOKALE („VOWELS“)

Das Vokaltrapez wird in XIPA durch eine Grafik im SVG-Format dargestellt. Die Grundstruktur der Grafik – das Koordinatensystem[1] und die unterschiedlichen Ebenen der Grafik sowie die Beschriftungen – werden im `body`-Tag des XSL-Stylesheets definiert. Nach der SVG-Deklaration folgen mehrere Ebenen, die den Rahmen bestimmen, das Trapez aufspannen und letztendlich den Vokalraum füllen. Die einzelnen Ebenen befinden sich in Tags mit der Bezeichnung `g`. Elemente innerhalb des öffnenden und des schließenden `g`-Tags sind zu einer Gruppe zusammengefasst. Innerhalb der Gruppen werden die Elemente Pfad, Text und Kreis (`path`, `text` und `circle`) verwendet, die unterschiedliche Aufgaben übernehmen. Das Auffüllen des Vokaltrapezes mit Inhalten (Lauten) ist im darauffolgenden Beispiel in Abschnitt 3.3.2 beschrieben.

3.3.1 Grundgerüst der Vektorgrafik

Auszug aus dem body-Tag des Vokaltrapezes (Grundgerüst der Vektorgrafik)

```
<!-- ############################  SVG CHART VOWELS  ############################## -->
<svg
  xmlns="http://www.w3.org/2000/svg"    <!-- Deklaration des SVG-Namensraums -->
  version="1.1"                         <!-- Angabe der SVG-Version bzw. Spezifikation -->
  width="465"                           <!-- Angabe der relativen Breite -->
  height="280"                          <!-- Angabe der relativen Höhe   -->
  viewBox="0 0 465 280">                <!-- Angabe des sichtbaren Bereichs (Min-x, Min-y Breite, Höhe) -->

  <!--############# MAIN FRAME ##############-->
  <g id="description">       <!-- <g>-Gruppe zur Beschriftung von Zungenposition und Kieferöffnungsgrad -->
   <!--DESCRIPTION TEXT-->
   <!-- Text-Elemente zur Ausgabe der Beschriftung. Die Position wird durch die Werte x und y bestimmt  -->
   <text x="83" y="10" text-anchor="middle" xsl:use-attribute-sets="vowels.text">Front</text>
   <text x="260" y="10" text-anchor="middle" xsl:use-attribute-sets="vowels.text">Central</text>
    ...
     <!--########### QUADRILATERAL #############-->  <!-- Aufspannen des Vokaltrapezes (Verhältnis 3:2) -->
     <!--  <g>-Gruppe des Trapezgerüsts - Ausgabe von horizontalen/vertikalen Linien und Ankerpunkten   -->
     <g id="quadrilateral" transform="translate(84,34)"> <!-- Verschieben aller folgenden Gruppierungen -->
        <!--VERTICAL LINES-->       <!-- durch die Verschiebung liegt der Ursprung des Trapezes bei 0:0 -->
        <!-- Pfade zur Darstellung der vertikalen und schrägen Linien des Trapezes                    -->
        <path d="M0   0 176   234" stroke-width=".3"/>  <!-- erste vertikale Linie (von i bis a)        -->
        <path d="M176 0 263.5 234" stroke-width=".3"/>
         ...
        <!--HORIZONTAL LINES-->
        <!-- Pfade zur Darstellung der horizontalen Linien (Koordinatenangaben und Strichstärke)        -->
        <path d="M0   0   351 0" stroke-width=".3"/>  <!-- erste horizontale Linie (von i bis u)        -->
        <path d="M59  78  351 78" stroke-width=".3"/>
         ...
        <!--########### ANCHORPOINTS ##############-->
        <!-- Koordinaten und Radius der Ankerpunkte (gefüllte schwarze Kreiselemente im Trapez)         -->
        <!--CLOSE-->
        <circle cx="0" cy="0" r="2.5"/>  <!-- 1. Punkt im Ursprung des Koordinatensystems (zw. i und y) -->
        <circle cx="176" cy="0" r="2.5"/>
         ...
          <!--########### VOWELSPACE ##############-->
          <g id="vowelspace">   <!-- Erstellen einer Gruppe für das Auffüllen des Trapezes mit Inhalten -->
            <xsl:call-template name="Vowels"/> <!-- Aufruf des Vowels-Templates zur Auswahl der Inhalte -->
          </g>
      </g>
    </g>
</svg>
```

1 Das Koordinatensystem hat in SVG-Grafiken seinen Ursprung (0, 0) am linken oberen Rand der Grafik und spannt sich nach unten hin auf. Die Koordinaten der Vokale im Vokaltrapez sind in Abbildung 1 dargestellt.

3.3.2 Das Vowels-Template

Das folgende Template ist modular aufgebaut, um eine spätere Integration sprachspezifischer Vokalräume zu ermöglichen. Das Template bedient sich in der Standardausführung an den in einer weiteren XSL-Datei hinterlegten x- und y-Koordinaten für die Position der jeweiligen Vokale. Die Koordinaten sind in der Datei `vowels.xsl` in Form von Attribut-Sets hinterlegt. Diese „internationale Variante“ gruppiert die Laute um die jeweiligen Ankerpunkte herum.

Die zur Anordnung verwendeten Positionen sind in Attribut-Sets hinterlegt, deren Bezeichnung eine Kombination des Präfix „int-“ und der Kardinalzahl des Vokals ist (vgl. Abb. 1 und Abb. 2). Die Attribut-Sets bestehen aus zwei Attributen mit den Namen `x` und `y`. In diesen sind die x- und y-Koordinaten der jeweiligen Laute gespeichert (z.B. im Attribut-Set `int-301` die Werte `x=-20` und `y=0` für den Vokal i).

Das Vowels-Template

```
<!-- ##########################  TEMPLATE VOWELS  ############################## -->
<xsl:template name="Vowels">
  <!-- Auswahl aller nicht-leeren Vokal-Elemente zur weiteren Verarbeitung im Template -->
  <xsl:for-each select="/ipa/sound[@type='vowel' and class!='emp']">
    <!-- Erzeugung der Variablen 'setname', Zusammensetzung aus dem Präfix 'int-' und der Kardinalzahl  -->
    <xsl:variable name="setname" select="concat('int-',name/cardinal/text())" />
    <!-- Die Variable 'set' sucht aus der Datei vowels.xsl das zum 'setname' passende Attribut-Set aus  -->
    <xsl:variable name="set" select="document('./vowels.xsl')/*/xsl:attribute-set[@name = $setname]" />

    <circle xmlns="http://www.w3.org/2000/svg" class='pos'><!-- erzeugt ein Kreiselement der Klasse pos -->
       <xsl:for-each select="$set/xsl:attribute"><!-- für jedes Attribut-Set aus 'vowels.xsl' werden: -->
         <xsl:attribute name="r">13</xsl:attribute>        <!-- der Radius des Kreis-Elements definiert -->
         <xsl:attribute name="c{@name}"> <!-- die Attribut-Namen von x und y in 'cx' und 'cy' gewandelt -->
           <xsl:value-of select="." />   <!-- die Werte von x/y für die Kreis-Attribute cx/cy gewählt   -->
         </xsl:attribute>
       </xsl:for-each>
         <xsl:call-template name="Transcribe">    <!-- Aufruf des Transcribe-Templates im Kreis-Element -->
           <xsl:with-param name="name" select="./name" />           <!-- Weitergabe des name-Parameters -->
         </xsl:call-template>
     <!-- durch den internen Aufruf von Add_metadata werden die Kreis-Elemente mit Metadaten versehen   -->
     <!-- die aktiven (klickbaren) Kreis-Elemente sind in Abb.2 durch orangene Umrandung dargestellt    -->
     </circle>

     <text xmlns="http://www.w3.org/2000/svg">
       <xsl:for-each select="$set/xsl:attribute"> <!-- erneute Auswahl der Attribut-Sets -->
         <xsl:attribute name="pointer-events">none</xsl:attribute>       <!-- deaktiviert 'Maus-Events' -->
           <xsl:attribute name="{@name}">          <!-- <text>-Elemente sind damit quasi 'durchsichtig' -->
             <xsl:choose>                          <!--  -->
               <xsl:when test="@name = 'y'">       <!-- Auswahl des Attributs mit dem Namen y           -->
                 <xsl:value-of select=". + 4" /><!-- Manipulation des y-Werts (horizontaler Ausgleich)  -->
               </xsl:when>
               <xsl:otherwise>     <!-- wenn @name (Name des Attributs) nicht y ist, dann ist @name = x -->
                 <xsl:value-of select="." /><!-- Auswahl des unveränderten x-Werts aus dem Attribut-Set -->
               </xsl:otherwise>
             </xsl:choose>
           </xsl:attribute>
       </xsl:for-each>
        <xsl:call-template name="Plot_Symbol">                  <!-- Aufruf des Templates Plot_Symbol -->
          <xsl:with-param name="name" select="./name"></xsl:with-param><!-- Ausgabe der Unicode-Zeichen -->
        </xsl:call-template>
     </text>

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

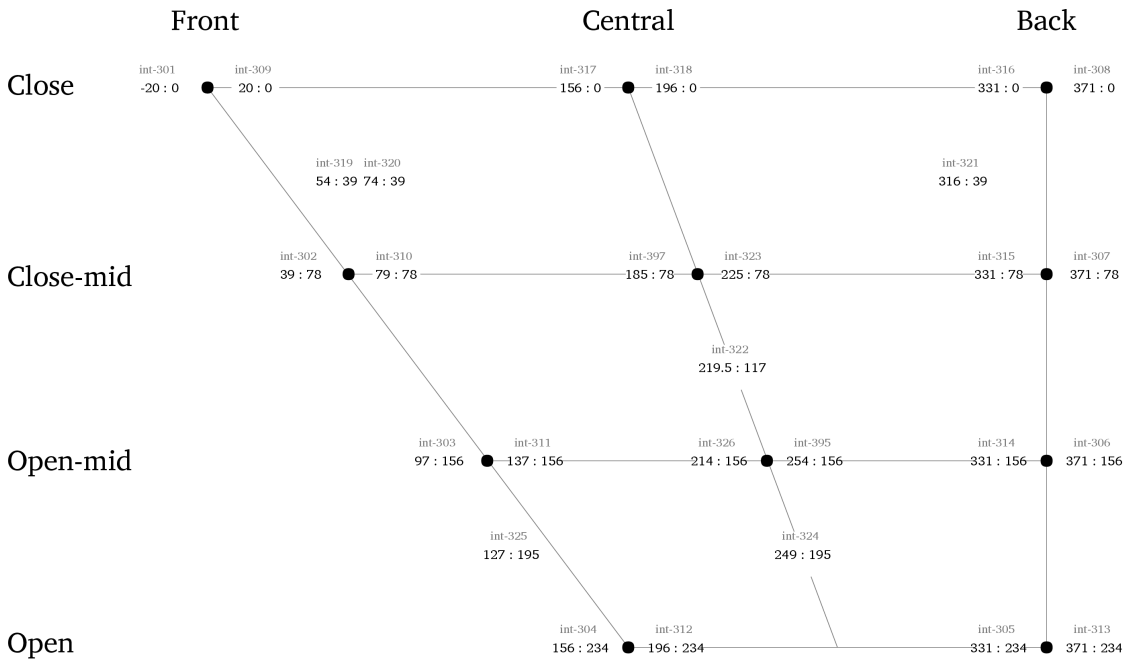

Abbildung 1: IDs und Koordinaten der Vokale im Vokaltrapez (Notation der Koordinaten: x : y)

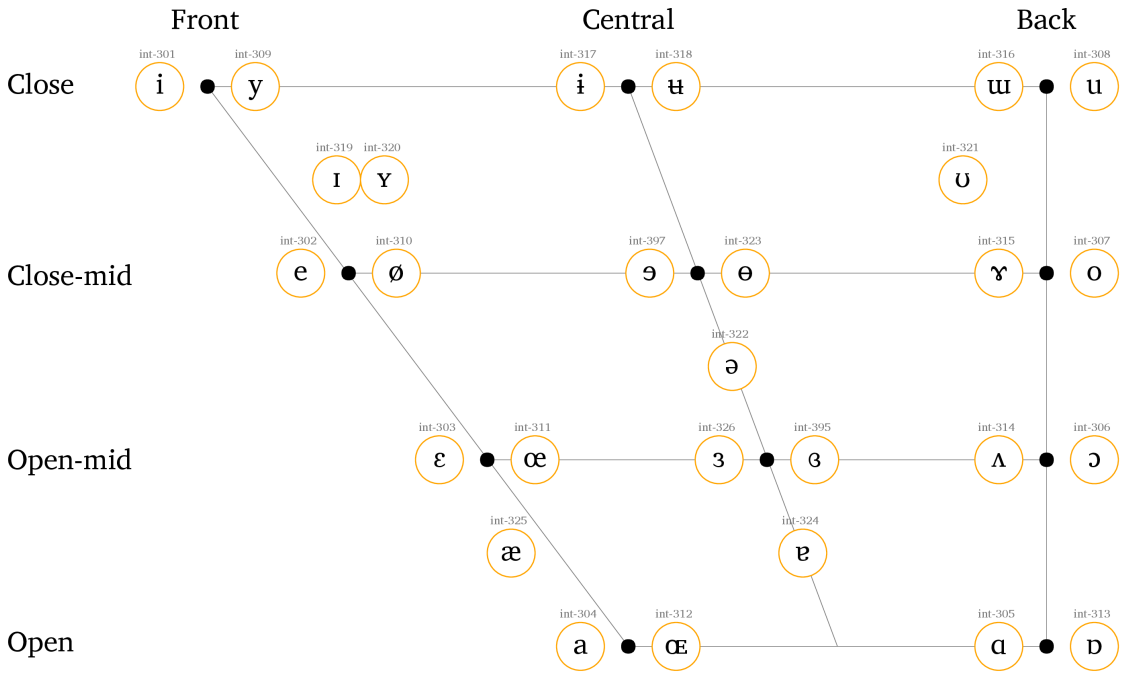

Abbildung 2: IDs und Circle-Elemente der Grafik (IDs entsprechen der Kardinalzahl der Vokale)

4 ANMERKUNGEN & WORKAROUNDS

4.1 XML

ESCAPE CHARACTERS Da einige Zeichen in XML bzw. XSLT reserviert sind (z.B. `\t` für `tabulator` und `\n` für `newline`) wurde an den betreffenden Stellen in der XML-Datei ein nullbreites Leerzeichen eingefügt, um die Verarbeitung des Kommandos zu verhindern. Dies trifft insbesondere auf einige LaTeX-Sequenzen und einige Kürzel der Praat-Notation zu. Das nullbreite Leerzeichen (`&#x200B;`) ist in der Ausgabe nicht sichtbar. Es macht sich durch einen zusätzlichen „Leerschritt“ bemerkbar, wenn der Cursor im Eingabefeld über eines der betroffenen Zeichen bewegt wird.

BACKSLASH Da die XSLT-Vorverarbeitung anführende Backslashs als Initialisierung einer Sequenz behandelt, sind in der Datei IPA.xml an diesen Positionen doppelte Backslashs eingegeben. Diese werden bei der Verarbeitung durch die XSL-Transformation automatisch entfernt.

4.2 XSLT

Bei der Transformation der XML-Datei durch XSLT werden Unicode-Entities direkt übersetzt. Da aus diesem Grund nicht auf die Entity-Referenz zugegriffen werden kann, sind die vierstelligen Kodes in einem weiteren Knoten des Laut-Elements hinterlegt. Für die Ausgabe des vollständigen Unicode-Kodes werden diese Referenzen (z.B. `0294`) in einem der Templates weiterverarbeitet und in die richtige Form (z.B. `&#x294;`) überführt.

4.3 SKRIPTE

Der Hinweis beim Versuch nicht existierende oder defekte Audiofiles abzuspielen wird durch einen simplen JavaScript `alert` ausgegeben. Um zu verhindern, dass der Nutzer bei jedem „Fehlversuch“ den Hinweis mit einem Klick auf den Ok-Button bestätigen muss, ist eine unaufdringlichere Lösung (z.B. in Form eines timergesteuerten Hinweisfeldes mit alertify.js etc.) wünschenswert.

4.4 BENENNUNG DER AUDIOFILES

Die Audiodateien in XIPA sind nach der im `entity`-Tag hinterlegten vierstelligen hexadezimalen Unicode-Referenz des Zeichens benannt (z.B. `0294.wav` für den Glottal Stop). Die-

se Dateinamenskonvention wurde gewählt, um eine robuste, betriebssystemübergreifende Verarbeitung der Dateien zu ermöglichen.

Theoretisch sind Dateinamen möglich, die dem Unicode-Zeichen entsprechen (z.B. ʔ.wav). Diese Lösung ist jedoch fragil. Sie führt zu Fehlern, insbesondere dann, wenn Dateien zwischen unterschiedlich formatierten Datenträgern kopiert werden sollen oder auf unterschiedlichen Plattformen genutzt werden.

4.5 WEB-AUDIO-API

Der `Audio-Context` der Web-Audio-API wird direkt zu Beginn des `onclickSwitcher` Skripts aufgerufen und ist damit für die gesamte Nutzungsdauer des Tools aktiv. In einigen Browsern kann dies zu einer erhöhten Last führen und bei Mobilgeräten zu einer verkürzten Akkulaufzeit führen. Dies trifft auch zu, wenn die Transkriptionsumgebung im Hintergrund ausgeführt wird.

Die Einbindung am Anfang des Skripts war notwendig, da die Web-Audio-API in verschiedenen Browsern unterschiedlich implementiert ist. Die eigentlich korrekte Vorgehensweise – das Öffnen und Schließen des Audio-Kontexts bei An- und Abwählen des `Play Sound`-Schalters – führte bei einigen Browsern (z.B. Mozilla Firefox) zu einer Fehlermeldung nach dem Abspielen von mehr als 6 Lauten, da der Audio-Kontext nicht – wie erwartet – geschlossen wurde.

Es ist zu erwarten, dass nachfolgende Browser-Versionen die API besser einbetten. Da es sich bei der API um einen Entwurf des W3C handelt, bleiben zukünftige Entwicklungen abzuwarten. Sobald ein Öffnen und Schließen des Audio-Kontexts im Skript browserübergreifend funktioniert, sollte diese Funktion entsprechend abgeändert werden.

4.6 NUTZEREINGABEN

In einigen Browsern bleiben eingegebene Transkriptionen auch bei einem erneuten Laden der Seite erhalten. Um diese Funktion browserübergreifend verfügbar zu machen und einen Verlust von Transkriptionen bei Computerabstürzen etc. zu verhindern, kann entweder ein Cookie beim Nutzer gespeichert werden, das die Inhalte der zuletzt eingegebenen Zeichen im Textfeld enthält. Alternativ ist eine Umsetzung mit HTML5 `localStorage` ebenfalls denkbar.