



# **GUIDO to XML**

## **Einbettung des GUIDO Formats in ein XML konformes Modell**

Dokumentation des Praktikums  
am Fachgebiet Computermusik des Fachbereichs Informatik der  
Technischen Universität Darmstadt

im Wintersemester 2000/01

**Heiko Merten**  
Matr.Nr.: 1039410  
[heiko@avantgarde.de](mailto:heiko@avantgarde.de)

**Sven Simon**  
Matr.Nr.: 1039119  
[ssimon@gmx.de](mailto:ssimon@gmx.de)

**Andreas Berndt**  
Matr.Nr.: 1025343  
[berndt-andreas@t-online.de](mailto:berndt-andreas@t-online.de)

## Inhaltsverzeichnis

Seite

<b>1.Einleitung.....</b>	<b>4</b>
<b>2.Warum XML?.....</b>	<b>5</b>
<b>3.Konzeption des XML-konformen Modells.....</b>	<b>6</b>
<b>4.Die möglichen Umsetzungen des Konzepts.....</b>	<b>8</b>
4.1.XML-übliche Baumstruktur mit Hilfe der „elementList“.....	8
4.2.Indizierte Start- und Ende-Tags.....	9
4.3.Verwendung des ANY-Parameters.....	9
<b>5.Vergleich GUIDO vs. XML.....</b>	<b>11</b>
5.1.Umfang und Aufwand der Notation beider Sprachen.....	11
5.2.Erweiterbarkeit.....	11
<b>6.Anwendungsmöglichkeiten.....</b>	<b>12</b>
<b>7.Dokumentation der GUIDO-XML-Tags.....</b>	<b>13</b>
5.0.Standardelemente.....	13
7.2.Elemente der „basics.dtd“.....	14
7.2.1.Überblick.....	14
7.2.2.Erläuterungen und Zuordnung zu GUIDO	
.....	14
7.3.Elemente der „dynamics.dtd“.....	16
7.3.1.Überblick.....	16
7.3.2.Erläuterungen und Zuordnung zu GUIDO.....	16
7.4.Elemente der „tempos.dtd“.....	18
7.4.1.Überblick.....	18
7.4.2.Erläuterungen und Zuordnung zu GUIDO.....	18
7.5.Elemente der „slurs.dtd“.....	20
7.5.1.Überblick.....	20
7.5.2.Erläuterungen und Zuordnung zu GUIDO.....	20
7.6.Elemente der „notevary.dtd“.....	22
7.6.1.Überblick.....	22
7.6.2.Erläuterungen und Zuordnung zu GUIDO.....	22
7.7.Elemente der „misc.dtd“.....	23
7.7.1.Überblick – Teil1.....	24
7.7.2.Erläuterungen und Zuordnung zu GUIDO.....	24
7.7.3.Überblick – Teil2.....	26
7.7.4.Erläuterungen und Zuordnung zu GUIDO.....	26
7.7.5.Überblick – Teil3.....	28
7.7.6.Erläuterungen und Zuordnung zu GUIDO.....	28
<b>8.Implementierung GUIDO2XML.....</b>	<b>31</b>
8.1.Entwicklung des Konzepts.....	31
8.2.Recherche.....	32
8.3.Technische Realisation.....	32
8.3.1.Verzeichnisstruktur.....	32
8.3.2.Filehandling.....	33
8.3.3.Stackhandling.....	33

8.3.4.Taghandling.....	33
8.3.5.Mehrdeutige Tags.....	33
8.3.6.Typabhängige Tags.....	33
8.3.7.Tag-Rule-Liste.....	34
8.4.Dokumentation.....	34
<b>9.Implementierung XML2GUIDO.....</b>	<b>35</b>
9.1.Programmbenutzung.....	35
9.2.Vorgehensweise des Programms.....	35
9.3.Erweiterbarkeit.....	36
<b>10.Literaturverzeichnis.....</b>	<b>37</b>

## 1. Einleitung

Ende der 90er Jahre wurde im Fachgebiet Computermusik des Fachbereichs Informatik der TU Darmstadt unter der Leitung von Herrn Hoos die musikalische Notationssprache GUIDO entwickelt. Für die heutige Zeit der weltweiten Vernetzung und des rasant wachsenden Datentransfers lässt sich der hohe Nutzen einer textbasierten und daher sehr komprimierten und sparsamen Form der Repräsentation von Musikstücken leicht erkennen.

Grund für die Entwicklung und Implementierung dieser neuen Notensprache neben den bereits bestehenden, universell anerkannten Standards wie z.B. MIDI war neben der Absicht einer schlankeren Programmierung besonders die Erkenntnis, dass von den bestehenden Formaten eine ganze Reihe musikalischer Details nicht oder nur unzureichend erkannt bzw. verarbeitet werden können.

Um die Popularität und das Interesse an GUIDO weiter zu steigern, ist in besonderem Maße die Möglichkeit von Bedeutung, Daten von bestehenden Lösungen in die GUIDO-Sprache zu konvertieren, bzw. umgekehrt.

Besonders interessant erschien in diesem Zusammenhang, eine Verbindung zwischen GUIDO und dem in seiner Bedeutung rasant wachsenden Standard der XML-Sprachen zu schaffen.

Darauf basierte die Aufgabenstellung unseres Praktikums:

- Die Konzeption eines XML-konformen Datenformats, das alle Merkmale von GUIDO repräsentieren kann.
- Die Implementierung eines Übersetzers, der sowohl GUIDO-Dateien in das neu entwickelte Format transferieren, als auch in umgekehrter Richtung die XML-konformen Musikdaten in das GUIDO Format umwandeln kann.

Das Praktikum fand im Wintersemester 2000/2001 unter der Leitung und Betreuung von Dipl.-Inform. Kai Renz statt.

.

## 2. Warum XML?

Der Hauptgesichtspunkt für die Entscheidung, eine Schnittstelle zwischen der GUIDO-Notationssprache und den allgemeinen XML-Sprachen zu schaffen, war sicherlich die ungebremst schnelle Verbreitung dieser neuen Art der Markup Sprachen. XML gestaltete sich zunächst als Kompromisslösung zwischen dem unübersichtlichen überladenen SGML-Ansatz und der zwar erfolgreichen, aber für viele Anwendungen, die über die einfache Webseitenerstellung hinausgehen, nicht ausreichenden Hypertext Markup Language (HTML). Dabei heraus kam schließlich ein Konzept, das auf wenigen grundlegenden Vorschriften basiert (→ Definition der Wohlgeformtheit) und für jede Anwendung individuell angepasst werden kann.

Durch die Entwicklung einer Verbindung zu XML, bzw. einer darauf aufbauenden XML konformen Lösung (GUIDO-XML) wird auch die GUIDO-Notationssprache von der weiten Verbreitung des neuen Standards profitieren können. Diese Aufwertung verwirklicht sich in der Folge darin, ein weiteres Argument für den Einsatz und Umstieg auf GUIDO als Format zur Repräsentation von Musikstücken anführen zu können.

XML verwendet wie alle Markup Sprachen eine strenge, ineinander geschachtelte Blockstruktur. Öffnende und schließende Tags, die nach Belieben benannt werden können, begrenzen die einzelnen Bereiche. Daneben gibt es zusätzlich die Möglichkeit, sogenannte „Leere Blöcke“ zu verwenden, die lediglich mit Attributen besetzt werden können und keinen Bereich umschließen.

Auch in der Musik spielen Blöcke und Bereiche eine ganz entscheidende Rolle. Seien das formelle dynamische Bezeichnungen wie *piano*, *crescendo*, *staccato*, *etc.* oder analytische Betrachtungen, die fast immer auf *Phrasen*, *Takten*, *Strophen*, *etc.* basieren. Überall findet sich die Blockstruktur, die sich geradezu anbietet, in einer XML konformen Sprache abgebildet zu werden.

Darüber hinaus liegt der musikalischen Notation wie auch den XML-Sprachen eine hierarchische Struktur zu Grunde, woraus bereits ersichtlich wird, dass ein Übergang zwischen beiden ohne Verfälschungen und hohe Komplexität realisierbar ist. Nach David Huron ist XML strukturell isomorph zur Musiknotationssprache, und genau das ist ja GUIDO.

### 3. Konzeption des XML-konformen Modells

XML ist ein neuer Standard der Markup-Sprachen, dessen wirklicher Einfluss heute noch gar nicht vollständig einzuschätzen ist. Ziel bei der Entwicklung war es unter anderem gewesen, die Transparenz und Lesbarkeit des Quellcodes durch die freie Wahl der Tag-Namen zu erhöhen. Außerdem lassen sich somit passende XML-konforme Ansätze für völlig unterschiedliche Anwendungen entwickeln, die gemeinsam lediglich auf dem sehr allgemein gefassten Regelwerk der Wohlgeformtheit von XML-Dateien basieren.

Unser Entwurf beinhaltet eine umfangreiche DocumentTypeDefinition (DTD), die nach den Regeln der Wohlgeformtheit ein fester Bestandteil eines XML-konformen Formats ist. Die DTD ermöglicht bereits zu einem sehr frühen Zeitpunkt die Überprüfung der Syntax des Dokuments, indem sie Regeln und Vorschriften für die unterschiedlichen Tags definiert.

Zur besseren Übersichtlichkeit und Gliederung und besonders auch im Hinblick auf spätere Erweiterungen entschieden wir uns dafür, die DTD in verschiedene Teilmodule zu zerlegen, die dann in einer Main-DTD (heißt bei uns „guido.dtd“) vereinigt werden. So ist es auch mit relativ niedrigem Aufwand möglich, bestimmte Merkmale aus dem Sprachumfang zu entfernen, indem einfach die Verknüpfung mit dem entsprechenden DTD-Modul aus der Main-DTD entfernt wird.

Wir entschieden uns für die folgenden DTD-Module:

- Basics.dtd  
Grundlegende musikalische Elemente wie Noten, Akkorde und Pausen
- Tempos.dtd  
Möglichkeiten zur Variation der Tempi
- Dynamics.dtd  
Möglichkeiten zur dynamischen Variation
- Slurs.dtd  
Bindungen, Haltebögen
- Notevary.dtd  
Angaben zum klanglichen Charakter einzelner Noten
- Misc.dtd  
sonstige musikalische Funktionen
- Advanced.dtd  
erweiterte Funktionen zur Formatierung und Darstellung aus „Advanced GUIDO“

Aus der Tatsache, dass seit der Entwicklung von XML noch nicht viel Zeit vergangen ist und es für den privaten Gebrauch praktisch noch keine XML-Anwendungen gibt, ergaben sich für uns zunächst Probleme, die ersten Entwürfe von GUIDO-XML zu testen.

Die üblichen Internet-Browser unterstützen XML erst in den aktuellsten Versionen (Explorer ab Version 5.0, Netscape ab Version 6.0) und dort meistens auch nur eingeschränkt.

Beispielsweise akzeptierte keiner der Browser das laut XML-Konvention vereinbarte Format für Kommentare:

```
% hallo, ich bin ein Beispielkommentar
```

Weiterhin wurden Regelvorgaben in den DTDs teilweise nicht berücksichtigt.

- Die REQUIRED-Eigenschaft für Tag-Attribute, also die Vorgabe, dass bestimmte Attribute mit Werten besetzt sein müssen, wurde nicht beachtet.
- Keiner der Browser bemängelte fehlerhaft benannte, bzw. in der DTD nicht aufgeführte Tag-Namen. Somit wäre es möglich, beliebige Tags hinzuzufügen, und genau das soll ja durch die DTD verhindert werden.

Aufgrund dieser grundlegenden Probleme, begannen wir damit, eine alternative Möglichkeit zur Überprüfung der XML-Konformität zu suchen und entschieden uns für XMLSPY, einer Entwicklungs- und Testumgebung für XML-konforme Sprachen.

## **4. Die möglichen Umsetzungen des Konzepts**

### **4.1. XML-übliche Baumstruktur mit Hilfe der „elementList“**

Bei der ersten Implementierung der DocumentTypeDefinition für GUIDO-XML erreichten wir schon bald den ersten Fall, für dessen Umsetzung wir bei der Übersetzung nach GUIDO einen Kompromiss eingehen mussten.

Innerhalb der musikalischen Notation gibt es eine ganze Reihe von Möglichkeiten bestimmte Blöcke von Noten in Lautstärke, Tempo oder klanglichem Charakter zu verändern. Diese sind folgerichtig ebenfalls in GUIDO abgebildet, lassen sich aber nicht problemlos in eine wohlgeformte XML-Sprache überführen. Theoretisch ist es möglich beispielsweise innerhalb eines crescendos eine im Umfang unbegrenzte Kombination aus Noten, Akkorden, Pausen, u.v.m. unterzubringen. Das gleiche gilt für dynamische Anweisungen und sogar für einfache Bindebögen.

Entsprechend der XML-Spezifikation müssten innerhalb der DTD alle möglichen „Child“-Elemente mit ihrer jeweiligen Häufigkeitsbeschränkung aufgeführt werden. Das führt im vorliegenden Fall allerdings zu einer sehr umfangreichen Aufzählung möglicher Elemente, die wir in der „elementList“ zusammengefasst haben. Die „elementList“ stellt ein „Parameter-Entity“ dar, das auf der obersten DTD-Ebene definiert wird. Alle folgenden DTD-Module können somit immer dann auf die elementList zugreifen, wenn innerhalb eines Element-Tags beliebige Nachfolge-Elemente stehen können.

Dieses Konzept erspart viel Text, bewirkt aber im Endeffekt bei der Umsetzung und Überprüfung eines GUIDO-XML-Dokuments lediglich eine simple Textersetzung an den betreffenden Stellen. Ein weit bedeutenderer Vorteil ist – wie sich wenig später noch deutlicher herausstellen wird – die Erhaltung der für die Markup-Sprachen typischen Baumstruktur, die auch eine Kontrollfunktion beinhaltet (Element X darf/muss bestimmte „Child“-Elemente haben).

Der Nachteil liegt unübersehbar in der Notwendigkeit, bei jeder zukünftigen Änderung bestehender oder Hinzunahme neuer Elemente, die „elementList“ entsprechend anzupassen, da die DTD sonst die neuen Elemente innerhalb von Blöcken nicht akzeptiert.

Aus diesem Grund entschieden wir uns in Absprache mit Kai Renz dazu, parallel zu unserer ersten Implementierung eine weitere Strategie umzusetzen, die in der Folge auch die Programmierung einer zusätzlichen Variante des Übersetzers von GUIDO nach GUIDO-XML bzw. umgekehrt nach sich zog.



#### **4.2.Indizierte Start- und Ende-Tags**

Die alternative Idee zur Umgehung der „elementList“ bzw. der generelle Verzicht auf parametrische Platzhalter erwuchs aus dem Problem, sich gegenseitig überkreuzende Bindebögen in der DTD abzubilden. Die Wohlgeformtheit der XML-Sprachen verbietet die Überschneidung von zwei oder mehreren Tag-Blöcken.

Bsp.:

```
<kino>
  <kontrolleur1 name="Bruce Willis">
    <gast1/>
    <gast2/>
    <gast3/>
  <kontrolleur2 name="Arnold Schwarzenegger">
    <gast4/>
    <gast5/>
  </kontrolleur1>
    <gast6/>
    <gast7/>
  <film titel="Cast Away">
    <gast8 state="zu spät"/>
  </kontrolleur2>
    <gast9 state="kein Einlass mehr"/>
  </film>
</kino>
```

Eine Lösung des Problems ergab sich dadurch, dass die DTD für jedes dieser „Problem-Tags“ – und dazu kommen wie oben erwähnt auch alle diejenigen, die auf die „elementList“ zugreifen – zwei getrennte, voneinander unabhängige EMPTY-Tags enthält. EMPTY bedeutet, dass diese Tags alleine stehen, also keinen öffnenden und schließenden Teil haben. Somit kann man tatsächlich auf die „elementList“ verzichten, nimmt aber gleichzeitig in Kauf, dass nun Anfangs- und Endetags in beliebiger Reihenfolge im XML-Dokument erscheinen können, ohne dass die DTD eventuelle Fehler bemängelt.

Zur korrekten Übersetzung zwischen GUIDO und GUIDO-XML müssen diese Tags nun zusätzlich mit einem Attribut versehen werden, dass jedem eine eindeutige ID zuordnet. Nur so ist es möglich Anfangs- und Endetags einander korrekt zuzuordnen. Die ehemals von der DTD übernommenen Kontrollfunktionen können von einem entsprechend intelligenten Übersetzer erledigt werden.

Darüber hinaus erhielten einige Elemente ein weiteres boolsches Attribut, aus dem hervorgeht, ob der zugehörige GUIDO-Ausdruck mit oder ohne Klammerung auftrat. Dieser Fall ergibt sich beispielsweise für Bindebögen, crescendi und diminuendi, bei denen in GUIDO sowohl mit einer BEGIN- und END-Konstruktion als auch mit Klammerung gearbeitet werden kann, um den betroffenen Bereich zu markieren. Würde dieses Attribut nicht besetzt werden, wäre bei der Rückübersetzung von GUIDO-XML nach GUIDO nicht mehr zu erkennen, ob das betroffene Element ursprünglich mal geklammert oder in Anfangs- und Endetag unterteilt gewesen war. Information würde durch die Übersetzung verlorengehen.

Nach der Fertigstellung der DTD-Dateien stellte sich leider bald heraus, dass dieser Ansatz nur bedingt durchführbar ist. Die XML-Definitionen für Wohlgeformtheit schreiben vor, dass stets ein oberstes, alles umschließendes „root“-element angegeben werden muss. Das hat zur Folge, dass an dieser Stelle auch die möglichen Nachfolgeelemente, die sich unterhalb befinden können, deklariert werden müssen.

#### **4.3.Verwendung des ANY-Parameters**

Je mehr uns das soeben geschilderte Problem zunächst zu schaffen machte und uns fast dazu bewegte, diesen Ansatz völlig aufzugeben, um so einfacher gestaltete sich daraufhin die Lösung:

Tatsächlich gibt es nämlich bei der Elementdeklaration einen (wenn nicht den einfachsten) Platzhalter, der ausdrückt, dass zwischen dem Start- und dem Ende-Tag eines Elements alles mögliche erscheinen kann. So lässt sich das eben beschriebene Deklarationsproblem beim Ansatz mit indizierten Tags ganz einfach mit folgendem Ausdruck beheben:

```
<!ELEMENT guido ANY>
```

Aus dieser Erkenntnis erwuchs natürlich auch die Möglichkeit, das Konzept mit der `elementList` noch einmal zu überarbeiten und die entsprechenden Stellen, an denen darauf zugegriffen wurde (`slur`, `cresc`, `dim`, ...) einfach durch ANY-Deklarationen zu ersetzen.

Der große Vorteil dabei ist der vollständige Verzicht auf ein Konstrukt wie die `elementList`, das bei jeder Ergänzung bzw. Entfernung von Unterelementen entsprechend abgeändert werden muss. Ein Nachteil ergibt sich daraus, dass man auf eine Kontrollfunktion, die die `DocumentTypeDefinition` bietet, verzichtet und zwischen bestimmten Tags alles erlaubt.

```
<guido>
  <note name="f" duration="1/8"/>
  <irgendein_falscher_tag/>
  <noch_ein_falscher_tag>
  <rest duration="1/4"/>
  </noch_ein_falscher_tag>
</guido>
```

## **5.Vergleich GUIDO vs. XML**

### **5.1.Umfang und Aufwand der Notation beider Sprachen**

Bereits bei kurzer Betrachtung der im Rahmen der Sprachdokumentation (siehe Kap. 7) angegebenen Beispiele, lässt sich erkennen, dass GUIDO im Vergleich mit XML eine deutlich komprimiertere Form der textuellen Musik-Notation darstellt.

Durch die klar strukturierte Syntax müssen Noten beispielsweise nicht explizit als solche deklariert werden, sondern es genügt, den Notennamen (z.B. e) anzugeben. Darüber hinaus können außer dem Notennamen alle Attribute von der vorhergehenden Note durch einfaches Weglassen übernommen werden.

Die GUIDO-XML-Sprache hingegen benötigt für jede Note einzelne EMPTY-Tags, innerhalb denen die Attribute festgelegt werden müssen. Grob geschätzt benötigt dieser Ansatz den ca. 10fachen Speicherplatz.

Für die XML-Lösung spricht wiederum die übersichtliche Darstellung und Lesbarkeit. Das kommt durch die hierarchische Baumstruktur, die sich bis zu einem gewissen Grad – wie weiter oben schon erwähnt – auch in der musikalischen Notation wiederfindet.

Sicherlich liegt es im Bereich des Möglichen, eine Anwendung zu entwickeln, die nach Art eines WYSIWYG-Editors den umfangreichen XML-Code im Hintergrund generiert. Nicht zu vergessen ist aber, dass das Speichervolumen, das die GUIDO- bzw. GUIDO-XML-Dateien benötigen stark unterschiedliche Ausmaße annehmen wird, und das ist für Internetanwendungen ein entscheidendes Kriterium.

Der GUIDO2XML-Konverter liefert somit das Verbindungsglied zwischen dem platzsparenden, komprimierten Ansatz von GUIDO und der hohen Funktionalität und weiten Verbreitung von XML.

### **5.2.Erweiterbarkeit**

Da die Entwicklung der GUIDO-Notationssprache noch nicht abgeschlossen ist, haben wir besonderen Wert auf die Möglichkeit der nachträglichen Erweiterung des Sprachumfangs gelegt.

So lässt sich auf XML-Seite die Document Type Definition einfach durch ein neues DTD-Modul erweitern, indem man in der „guido.dtd“ einen Verweis auf die neue DTD-Datei einfügt.

Betrachtet man den eigentlichen Übersetzer können entsprechende Änderungen zentral in der Datei „taglist.h“ vorgenommen werden. Hier werden sowohl Tagnamen als auch die möglichen Attribute vordefiniert. Weitere Erläuterungen zur „taglist“ finden sich in Kapitel 8.

## **6. Anwendungsmöglichkeiten**

Einige Mitarbeiter des betreuenden Fachgebiets haben sich bereits während der Konzeption und Entwicklung des Konverters Gedanken über dessen weitere praktische Nutzung gemacht. Die Ergebnisse sollen in naher Zukunft in einer wissenschaftlichen Abhandlung („XML and GUIDO – A winning combination“) zusammengefasst werden.

Demnach wird der Hauptnutzen des Konverters in der Zukunft wohl in den umfangreichen Analyse- und Ordnungsmöglichkeiten liegen, die die bereits existierenden auf XML basierenden Datenbank Anwendungen bieten.

Ein konkretes Vorhaben besteht zum Zeitpunkt der Entwicklung des Konverters jedoch noch nicht.

Aus unserer Sicht lässt sich hoffen, dass es in absehbarer Zeit eine konkrete Anwendungsmöglichkeit geben wird, für die die Ergebnisse unseres Praktikums relevant sein werden.

## 7. Dokumentation der GUIDO-XML-Tags

### Hinweise zur Notation

Die Dokumentation der einzelnen GUIDO-XML-Tags folgt immer dem selben Schema: Zunächst der Name des Tags. Handelt es sich um einen EMPTY-Tag, ist dies zusätzlich hinter dem anschließenden Doppelpunkt angegeben.

Bei zweigeteilten Tags, die nur bestimmte Nachfolgeelemente enthalten dürfen, sind diese Elemente explizit hinter dem Doppelpunkt aufgezählt. Ein sich daran anschließendes „+“-Zeichen besagt, dass das Nachfolgeelement mindestens einmal erscheinen muss.

Bei zweigeteilten ANY-Tags – also Tags, die keine Einschränkungen bzgl. der Nachfolgeelemente besitzen – ist nichts hinter dem Tag-Namen angegeben.

Tags, die es sowohl als EMPTY- als auch als zweigeteilte Version gibt sind durch ein „?“ zwischen Tagname und EMPTY gekennzeichnet.

### 5.0. Standardelemente

Die drei folgenden Tags sind in jedem GUIDO-XML Dokument enthalten. Insbesondere ist `<guido>` das „Root“-Element, das am Anfang und Ende des Dokumentes steht und alles umschließt. Jedes Dokument muss ein Segment enthalten, das wiederum mehrere Sequenzen (mindestens eine) umfasst.

#### **`<guido>:segment+`**

ist wie eben schon erwähnt das „Root“-Element, das die Definition der Wohlgeformtheit für XML konforme Sprachen vorschreibt.

#### **`<segment>:sequence+`**

kann man sich am besten als Notensystem vorstellen, das mehrere Einzelstimmen vereinigt. Die Stimmen können mit dem `<staff>`-Tag (s. 5.6.5) beliebig auf verschiedene Notensysteme verteilt werden.

#### **`<sequence>`**

ist eine einzelne Stimme, die Noten, Pausen, Akkorde, dynamische und Tempoelemente, u.v.m. beinhalten kann.

*Beispiel:*

```

<guido>
  <segment>
    <sequence>
      ....
    </sequence>
    <sequence>
      ....
    </sequence>
  </segment>
</guido>

```

*in GUIDO:*    { [ ... ] , [ ... ] }

## 7.2. Elemente der „basics.dtd“

### 7.2.1. Überblick

Elm	clef	EMPTY			
▲	clef attribute list				
		Att Name	Att Type	Att Values	Att Presence
	1	name	CDATA		#REQUIRED
Elm	key	EMPTY			
▲	key attribute list				
		Att Name	Att Type	Att Values	Att Presence
	1	name	CDATA		#IMPLIED
	2	number	CDATA		#IMPLIED
Elm	meter	EMPTY			
▲	meter attribute list				
		Att Name	Att Type	Att Values	Att Presence
	1	name	CDATA		#REQUIRED
Elm	note	EMPTY			
▲	note attribute list				
		Att Name	Att Type	Att Values	Att Presence
	1	name	CDATA		#REQUIRED
	2	octave	CDATA		#IMPLIED
	3	duration	CDATA		#IMPLIED
	4	accidentals	CDATA		#IMPLIED
Elm	rest	EMPTY			
▲	rest attribute list				
		Att Name	Att Type	Att Values	Att Presence
	1	duration	CDATA		#IMPLIED
▲	chord choice of				
	Elm	rest	1 or more		
	Elm	note	1 or more		

### 7.2.2. Erläuterungen und Zuordnung zu GUIDO

#### <clef>:EMPTY

bezeichnet wie bei GUIDO den Notenschlüssel. Dieser Tag kann innerhalb eines Stückes beliebig oft gewechselt werden. Das Element enthält als einziges Attribut den Namen des Schlüssels, der unbedingt angegeben werden muss. Insgesamt ist das Element „clef“ in einem GUIDO-XML Dokument nicht zwingend erforderlich. Die Benennung der unterschiedlichen Schlüssel ist analog zu GUIDO.

Beispiel:     <clef name="g2"/>  
in GUIDO:     \clef<"g2-8">

#### <key>:EMPTY

definiert die Tonart des Stückes. Das erforderliche Attribut „name“ beinhaltet den Namen der Tonart, der sich an die Konvention aus GUIDO hält. Alternativ dazu kann auch unter „number“ die Anzahl der Vorzeichen angegeben werden: positive Zahlen für Kreuze, negative für b's.

Beispiel:     <key name="a#"/>

in GUIDO: `\key<"a#">`

### **<meter>:EMPTY**

legt die Taktart des Stücks fest. Die Attributbezeichnung ist obligatorisch, kann aber innerhalb des Stückes beliebig oft wechseln.

Beispiel: `<meter name="4/4"/>`

in GUIDO: `\meter<"4/4">`

### **<note>:EMPTY**

beinhaltet mehrere Attribute wie „octave“ oder „duration“ (siehe Übersicht), jedoch nur der „name“ ist verpflichtend anzugeben. Die Bezeichnungsweise ist erneut konform zu der von GUIDO, Notenwerte können aber im Gegensatz dazu von der Vorgängernote nicht durch einfaches Weglassen übernommen werden, sondern müssen explizit angegeben werden. Eventuelle Punktierungen lassen sich zusammen mit der Dauer berücksichtigen (duration="1/2."). Wie bei GUIDO existieren aber auch hier Defaultwerte für Oktave oder Dauer, die beim Auslassen verwendet werden. Vorzeichen können im Attribut „accidentals“ eingetragen werden (Bsp.: „#“, „bb“, „###“). Das „&“-Zeichen aus der GUIDO-Notation kann in XML nicht verwendet werden, da es als Steuer- und Sonderzeichen behandelt wird.

Beispiel: `<note name="d" octave="1" duration="3/4"/>`

in GUIDO: `d1*3/4`

### **<rest>:EMPTY**

erzeugt eine Pause bestimmter Länge. Wie auch bei den Noten kann die Punktierung im Duration-Attribut angegeben werden.

Beispiel: `<rest duration="1/4"/>`

in GUIDO: `_/4`

### **<chord>**

definiert Akkorde aus den Noten, die vom Anfangs- und Endetag umschlossen werden. Die Längen der einzelnen Noten können dabei unterschiedlich sein. Die Anzahl der übereinandergeschichteten Noten ist unbegrenzt.

Beispiel: `<chord>`  
           `<note name="e" octave="0"/>`  
           `<note name="e" octave="1"/>`  
           `<note name="g" octave="1" accidentals="#"/>`  
           `<note name="a" octave="1"/>`  
           `</chord>`

in GUIDO: `{e0*1/4, e1*1/4, g#, a}`

## 7.3. Elemente der „dynamics.dtd“

### 7.3.1. Überblick

Elm intens	EMPTY			
intens attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	dynamic	CDATA		#IMPLIED
2	real_intens	CDATA		#IMPLIED
Elm i	EMPTY			
i attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	dynamic	CDATA		#IMPLIED
2	real_intens	CDATA		#IMPLIED
Elm cresc	ANY			
cresc attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	dynamic	CDATA		#IMPLIED
2	intensity	CDATA		#IMPLIED
Elm crescBegin	EMPTY			
crescBegin attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
2	dynamic	CDATA		#IMPLIED
3	intensity	CDATA		#IMPLIED
Elm crescEnd	EMPTY			
crescEnd attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
Elm dimin	ANY			
dimin attribute list				
Elm diminBegin	EMPTY			
diminBegin attribute list				
Elm diminEnd	EMPTY			
diminEnd attribute list				

### 7.3.2. Erläuterungen und Zuordnung zu GUIDO

#### <intens>:EMPTY

legt die Intensität der darauf folgenden Noten fest. Diese Angabe ist bis zum nächsten Auftreten eines „intens“-Tags gültig. „i“ ist die in GUIDO verwendete Kurzform, hat aber genau die selbe Funktionsweise.

Neben der Angabe der Lautstärke unter „dynamic“, kann analog zu GUIDO im Attribut „real\_intens“ eine relative numerische Veränderung der Intensität angegeben werden.

Beispiel:      <intens dynamic=“mf“/>

in GUIDO:    \intens<“mf“>

#### <cresc>?EMPTY



definiert ein crescendo über alle von den Tags umschlossenen Noten bis zu der Lautstärke, die im Attribut „dynamic“ angegeben ist. Wie bei „intens“ kann auch hier im gleichnamigen Attribut ein Zahlenwert angegeben werden, mit dem die Lautstärke relativ verändert wird.

Für Überkreuzungen von zwei crescendi gibt es ein zusätzliches Konstrukt, das aus einem **<crescBegin>** und einem **<crescEnd>** besteht. Beide Tags benötigen einen zueinander passenden Index um den Notenbereich klar angeben zu können, über den sich das crescendo erstreckt.

*Beispiel:*      <cresc dynamic="ff">  
                    <note name="a" octave="1" duration="1/4">  
                    <note name="g" octave="1" duration="1/4">  
                    <note name="c" octave="1" duration="1/4">  
                  </cresc>

*in GUIDO:*    \cresc<"ff">(a1\*1/4 g c)

## **<dimin>?EMPTY**

verhält sich genauso wie „cresc“. Es gibt ebenfalls das Hilfskonstrukt für den Fall, dass sich mehrere Notenblöcke überschneiden. Die Bezeichnungen der Attribute sind identisch.

## 7.4. Elemente der „tempos.dtd“

### 7.4.1. Überblick

Elm tempo	EMPTY			
tempo attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	value	CDATA		#IMPLIED
2	real_value	CDATA		#IMPLIED
Elm accel	ANY			
accel attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	tempo	CDATA		#IMPLIED
2	real_tempo	CDATA		#IMPLIED
Elm accelBegin	EMPTY			
accelBegin attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
2	tempo	CDATA		#IMPLIED
3	real_tempo	CDATA		#IMPLIED
Elm accelEnd	EMPTY			
accelEnd attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
Elm rit	ANY			
rit attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	tempo	CDATA		#IMPLIED
2	real_tempo	CDATA		#IMPLIED
Elm ritBegin	EMPTY			
ritBegin attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
2	tempo	CDATA		#IMPLIED
3	real_tempo	CDATA		#IMPLIED
Elm ritEnd	EMPTY			
ritEnd attribute list				
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED

### 7.4.2. Erläuterungen und Zuordnung zu GUIDO

#### <tempo>:EMPTY

wird – wie schon dem Namen nach – zur Festlegung bzw. Änderung des selbigen innerhalb eines Musikstücks verwendet. Als Parameter stehen die schriftliche Tempobezeichnung („Moderato“, „Allegro“, ...) und/oder ein absoluter Zahlenwert für die Länge einer bestimmten Notenart. Die Angaben gelten bis zur nächsten Tempoangabe.

Beispiel: <tempo value="Moderato" real\_value="1/4=60">

in GUIDO: \tempo<"Moderato","1/4=60">

#### <accel>?EMPTY

kennzeichnet einen definierten Notenbereich mit einem „accelerando“. Die betroffenen Noten werden von den Anfangs- bzw. Endetags umschlossen. Für sich überschneidende accelerandi oder Überkreuzungen mit anderen Notenbereichen gibt es auch die Möglichkeit, paarweise indizierte EMPTY-TAGs zu verwenden.

Wie beim eben beschriebenen Tempotag können nominale oder absolute Tempobezeichnungen über die Parameter festgelegt werden.

*Beispiel:*
















```
<accelBegin id="1"/>
    <note name="a" octave="0" duration="1/4"/>
    <note name="g" octave="0" duration="1/4"/>
    <note name="c" octave="1" duration="1/4"/>
<accelEnd value="Presto" real_value="1/2=160"/>
```

*in GUIDO:*    \accelBegin a0/4 g c1 \accelEnd<"Presto", "1/2=160">

**<rit>** verhält sich abgesehen von der musikalischen Bedeutung absolut identisch wie **<accel>**. Wir verzichten daher auf eine genauere Erklärung.

## 7.5. Elemente der „slurs.dtd“

### 7.5.1. Überblick

 <b>tie</b>	ANY																																								
 <b>tie attribute list</b>																																									
 <b>tieBegin</b>	EMPTY																																								
 <b>tieBegin attribute list</b>																																									
 <b>tieEnd</b>	EMPTY																																								
 <b>tieEnd attribute list</b>																																									
 <b>Comment</b>	Slurs																																								
 <b>slur</b>	ANY																																								
 <b>slur attribute list</b>																																									
<table><tr><th></th><th>Att. Name</th><th>Att. Type</th><th>Att. Values</th><th>Att. Presence</th></tr><tr><td>1</td><td>dx1</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>2</td><td>dy1</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>3</td><td>dx2</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>4</td><td>dy2</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>5</td><td>dx3</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>6</td><td>dy3</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr></table>			Att. Name	Att. Type	Att. Values	Att. Presence	1	dx1	CDATA		#IMPLIED	2	dy1	CDATA		#IMPLIED	3	dx2	CDATA		#IMPLIED	4	dy2	CDATA		#IMPLIED	5	dx3	CDATA		#IMPLIED	6	dy3	CDATA		#IMPLIED					
	Att. Name	Att. Type	Att. Values	Att. Presence																																					
1	dx1	CDATA		#IMPLIED																																					
2	dy1	CDATA		#IMPLIED																																					
3	dx2	CDATA		#IMPLIED																																					
4	dy2	CDATA		#IMPLIED																																					
5	dx3	CDATA		#IMPLIED																																					
6	dy3	CDATA		#IMPLIED																																					
 <b>sl</b>	ANY																																								
 <b>sl attribute list</b>																																									
 <b>slurBegin</b>	EMPTY																																								
 <b>slurBegin attribute list</b>																																									
<table><tr><th></th><th>Att. Name</th><th>Att. Type</th><th>Att. Values</th><th>Att. Presence</th></tr><tr><td>1</td><td>id</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>2</td><td>dx1</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>3</td><td>dy1</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>4</td><td>dx2</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>5</td><td>dy2</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>6</td><td>dx3</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr><tr><td>7</td><td>dy3</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr></table>			Att. Name	Att. Type	Att. Values	Att. Presence	1	id	CDATA		#IMPLIED	2	dx1	CDATA		#IMPLIED	3	dy1	CDATA		#IMPLIED	4	dx2	CDATA		#IMPLIED	5	dy2	CDATA		#IMPLIED	6	dx3	CDATA		#IMPLIED	7	dy3	CDATA		#IMPLIED
	Att. Name	Att. Type	Att. Values	Att. Presence																																					
1	id	CDATA		#IMPLIED																																					
2	dx1	CDATA		#IMPLIED																																					
3	dy1	CDATA		#IMPLIED																																					
4	dx2	CDATA		#IMPLIED																																					
5	dy2	CDATA		#IMPLIED																																					
6	dx3	CDATA		#IMPLIED																																					
7	dy3	CDATA		#IMPLIED																																					
 <b>slurEnd</b>	EMPTY																																								
 <b>slurEnd attribute list</b>																																									
<table><tr><th></th><th>Att. Name</th><th>Att. Type</th><th>Att. Values</th><th>Att. Presence</th></tr><tr><td>1</td><td>id</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr></table>			Att. Name	Att. Type	Att. Values	Att. Presence	1	id	CDATA		#IMPLIED																														
	Att. Name	Att. Type	Att. Values	Att. Presence																																					
1	id	CDATA		#IMPLIED																																					

### 7.5.2. Erläuterungen und Zuordnung zu GUIDO

#### <tie>

verbindet mehrere Noten gleicher Tonhöhe miteinander und vereinigt deren Tondauern. Die betreffenden Noten stehen zwischen den Anfangs- und Endtags von <tie>. Die optionalen Parameter für die Gestaltung des grafischen Verlaufs der Bindebögen sind in der GUIDO-Dokumentation erläutert.

Beispiel: <tie>

```
<note name="d" duration="1/4"/>
<note name="d" duration="1/16"/>
```

</tie>

*in GUIDO:* \tie(d1/4 d/16)

## <slur>?EMPTY


























erzeugt einen Bindebogen über die von den Tags umschlossenen Noten. Im Unterschied zum bereits erwähnten <tie> haben hier die Noten in der Regel unterschiedliche Tonhöhen. <slur> gibt es in Analogie zu GUIDO auch in der Kurzform <sl>, deren Funktionsweise identisch zu der von <slur> ist. Bei überkreuzenden Bindebögen können auch für <slur> indizierte EMPTY-Tags verwendet werden. Darüber hinaus können acht Parameter für die exakte Darstellung des Bindebogens als Bezier-Kurve verwendet werden.

*Beispiel:* <sl>  
          <note name="d" duration="1/4"/>  
          <note name="d" duration="1/16"/>  
          </sl>

*in GUIDO:* \sl(d1/4 d/16)

## 7.6. Elemente der „notevary.dtd“

### 7.6.1. Überblick

 <b>staccato</b>	ANY								
 <b>accent</b>	ANY								
 <b>tenuto</b>	ANY								
 <b>marcato</b>	ANY								
 <b>Comment</b>	Trills and Ornaments								
 <b>trill</b> sequence of	 <b>chord</b>								
 <b>trill</b> attribute list									
 <b>mordent</b> sequence of	 <b>chord</b>								
 <b>mordent</b> attribute list									
	<table><tr><th>Att. Name</th><th>Att. Type</th><th>Att. Values</th><th>Att. Presence</th></tr><tr><td>1 spec</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr></table>	Att. Name	Att. Type	Att. Values	Att. Presence	1 spec	CDATA		#IMPLIED
Att. Name	Att. Type	Att. Values	Att. Presence						
1 spec	CDATA		#IMPLIED						
 <b>turn</b> sequence of	 <b>chord</b>								
 <b>turn</b> attribute list									
 <b>Comment</b>	Tremolo								
 <b>tremolo</b>	ANY								
 <b>tremolo</b> attribute list									
 <b>Comment</b>	Fermatas								
 <b>fermata</b>	EMPTY								
 <b>fermataNotes</b>	ANY								
 <b>Comment</b>	Grace and Cue Notes								
 <b>grace</b>	ANY								
 <b>grace</b> attribute list									
	<table><tr><th>Att. Name</th><th>Att. Type</th><th>Att. Values</th><th>Att. Presence</th></tr><tr><td>1 spec</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr></table>	Att. Name	Att. Type	Att. Values	Att. Presence	1 spec	CDATA		#IMPLIED
Att. Name	Att. Type	Att. Values	Att. Presence						
1 spec	CDATA		#IMPLIED						
 <b>cue</b>	ANY								
 <b>cue</b> attribute list									
	<table><tr><th>Att. Name</th><th>Att. Type</th><th>Att. Values</th><th>Att. Presence</th></tr><tr><td>1 instrument</td><td>CDATA</td><td></td><td>#IMPLIED</td></tr></table>	Att. Name	Att. Type	Att. Values	Att. Presence	1 instrument	CDATA		#IMPLIED
Att. Name	Att. Type	Att. Values	Att. Presence						
1 instrument	CDATA		#IMPLIED						

### 7.6.2. Erläuterungen und Zuordnung zu GUIDO

#### 7.6.3.

**<staccato>**

**<accent>**

**<tenuto>**

**<marcato>**

Diese vier Tags weisen bestimmten Notenbereichen die jeweilige Ausdrucksform zu, die im Tagnamen angegeben ist. Attribute gibt es keine.

Beispiel: `<tenuto>`  
           `<note name="d" duration="1/4"/>`  
           `<note name="c" duration="1/4"/>`  
           `</tenuto>`

in GUIDO: `\ten(d1/4 c)`

**<trill>**

**<mordent> <turn>**

Zur klanglichen Variation von Akkorden können diese drei Tags verwendet werden. Das optionale Attribut „spec“ legt fest, in welcher Frequenz bspw. der Triller gespielt werden soll.

*Beispiel:*      <trill spec="32">  
                    <chord>  
                        <note name="f" duration="1/4"/>  
                        <note name="e" accidentals="b"/>  
                    </chord>  
                    </trill>  
*in GUIDO:*      \trill<32>({f/4,e&})

**<tremolo>**

Verhält sich ganz genauso wie eben zuvor beschriebenen Tags zur Akkordvariation, außer dass sich tremolo auf Noten und nicht auf Akkorde bezieht. Auch hier gibt das Attribut „spec“ an, wie die musikalische Bewegung des Tremolo gestaltet werden soll.

**<fermata...>?EMPTY**

setzt eine Fermate wahlweise auf einen Taktstrich – in diesem Fall ist der Fermata-Tag ein EMPTY-Tag – oder über eine oder mehrere Noten/Pausen. Der betroffene Bereich wird von den Anfangs- und Endetags umschlossen.

*Beispiel:*      <note name="e" duration="1/4"/>  
                    <fermata>  
                        <note name="d" duration="1/2"/>  
                    </fermata>  
*in GUIDO:*      e/4 \fermata(d/2)

**<grace> <cue>**

Diese Tags gleichen in der Anwendung völlig den am Anfang dieses Abschnitts beschriebenen Tags zur Änderung des Ausdrucks (marcato, staccato,...). Bei <cue> gibt es außerdem noch die Möglichkeit, explizit das Instrument anzugeben, das die gekennzeichneten Noten spielen soll.

*Beispiel:*      <cue instrument="Ob 1">  
                        <note name="d" duration="1/4"/>  
                        <note name="d" duration="1/4"/>  
                        <note name="d" duration="1/4"/>  
                    </cue>  
                        <note name="g" duration="1/2"/>  
*in GUIDO:*      \cue<"Ob 1">(d1/4 d d) g/2

**7.7.Elemente der „misc.dtd“**

### 7.7.1.Überblick – Teil1

Elm	instr	EMPTY		
▲	instr attribute list			
	Att. Name	Att. Type	Att. Values	Att. Presence
1	value	CDATA		#IMPLIED
2	value_s2	CDATA		#IMPLIED
Comment	Repetitions			
Elm	repeatBegin	EMPTY		
▲	repeatBegin attribute list			
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
2	times	CDATA		#IMPLIED
Elm	repeatEndNotes	EMPTY		
▲	repeatEndNotes attribute list			
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
2	times	CDATA		#IMPLIED
Elm	repeatEnd	EMPTY		
▲	repeatEnd attribute list			
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
Elm	repeatEndNotes...	ANY		
▲	repeatEndNotesEnd attribute list			
	Att. Name	Att. Type	Att. Values	Att. Presence
1	id	CDATA		#IMPLIED
2	times	CDATA		#IMPLIED
Comment	Octaver			
Elm	octave	EMPTY		
▲	octave attribute list			
	Att. Name	Att. Type	Att. Values	Att. Presence
1	transpose	CDATA		#IMPLIED
Elm	octaveNotes	ANY		
▲	octaveNotes attribute list			
	Att. Name	Att. Type	Att. Values	Att. Presence
1	transpose	CDATA		#IMPLIED

### 7.7.2.Erläuterungen und Zuordnung zu GUIDO

#### <instr>:EMPTY

definiert das Instrument, das eine bestimmte Stimme spielt. Es kann zu jedem Zeitpunkt und beliebig oft gewechselt werden und behält seine Gültigkeit bis zum nächsten Auftreten des <instr>-Tags. Wie bei GUIDO kann optional zur Bestimmung des Instruments zusätzlich noch ein diesem zugeordnetes angegeben werden (z.B. ein MIDI-Instrument).

Beispiel:     <instr value="violino 1"/>  
in GUIDO:     \instr<"violino 1">

#### <repeat...>?EMPTY

wird verwendet, um Abschnitte in bestimmter Weise zu wiederholen. Dafür stehen die folgenden Tags zu Verfügung:



<repeatBegin> und <repeatEnd>, bzw. <repeatEndNotes> und <repeatEndNotesEnd>, wobei die ersten drei indizierte EMPTY-Tags und der letzte ein normaler zweigeteilter Tag sind.

Der Zusatz „times“ des <repeatBegin>-Tags gibt an, wie oft der markierte Abschnitt wiederholt werden soll (Standard ist 2). Bei <repeatEndNotes> erklärt „times“, nach der wievielten Wiederholung der von diesem Tag umschlossene Bereich gespielt werden soll.

*Beispiel:*

```
<repeatBegin times="2"/>
  <note name="c" duration="1/2"/>
  <note name="g" duration="1/4"/>
  <note name="f" duration="1/4"/>
<repeatEndNotes times="4">
  <note name="e" duration="1/4"/>
  <note name="g" duration="1/2"/>
</repeatEndNotes>
```

*in GUIDO:*

```
\repeatBegin<2>
  c1/2 g/4 f
\repeatEnd<4>( e/4 g/2 )
```

## <octave...>?EMPTY

oktaviert die folgenden Noten in die im Attribut „transpose“ angegebene Oktave (n / -n / 0). Soll nur ein kurzer, fest abgegrenzter Bereich oktaviert werden, innerhalb dem sich keine weiteren zweigeteilten Tags befinden, deren Wirkungsbereich sich mit dem Oktavierungsbereich teilweise überschneidet, kann auch <octaveNotes> verwendet werden. Die Oktavierung gilt dann nur für die Noten innerhalb des Anfangs- und Endetags. Beim erstgenannten Fall gelten die Angaben bis zum nächsten Auftreten eines <octave>-Tags.







































*Beispiel:*

```
<note name="h" duration="2/4"/>
<octave transpose="1"/>
  <note name="c" octave="2"/>
  <note name="e" duration="1/2"/>
<octave transpose="0"/>
  <note name="e" duration="1/2"/>
```

*in GUIDO:*

```
h2/4 \oct<+1> c2 e/2 \oct<0> e1/2
```

### 7.7.3.Überblick – Teil2

 <b>text</b>	EMPTY			
▲ <b>text attribute list</b>				
	 <b>Name</b>	 <b>Type</b>	 <b>Values</b>	 <b>Presence</b>
	1 content	CDATA		#IMPLIED
	2 position	CDATA		#IMPLIED
 <b>t</b>	EMPTY			
▲ <b>t attribute list</b>				
	 <b>Name</b>	 <b>Type</b>	 <b>Values</b>	 <b>Presence</b>
	1 content	CDATA		#IMPLIED
	2 position	CDATA		#IMPLIED
 <b>comment</b>	EMPTY			
▲ <b>comment attribute list</b>				
	 <b>Name</b>	 <b>Type</b>	 <b>Values</b>	 <b>Presence</b>
	1 content	CDATA		#IMPLIED
 <i>Comment</i>	Add. Info			
 <b>title</b>	EMPTY			
▲ <b>title attribute list</b>				
	 <b>Name</b>	 <b>Type</b>	 <b>Values</b>	 <b>Presence</b>
	1 name	CDATA		#IMPLIED
 <b>composer</b>	EMPTY			
▲ <b>composer attribute list</b>				
	 <b>Name</b>	 <b>Type</b>	 <b>Values</b>	 <b>Presence</b>
	1 name	CDATA		#IMPLIED
 <i>Comment</i>	Labels and Marker			
 <b>mark</b>	EMPTY			
▲ <b>mark attribute list</b>				
	 <b>Name</b>	 <b>Type</b>	 <b>Values</b>	 <b>Presence</b>
	1 name	CDATA		#IMPLIED
 <b>label</b>	EMPTY			
▲ <b>label attribute list</b>				
	 <b>Name</b>	 <b>Type</b>	 <b>Values</b>	 <b>Presence</b>
	1 name	CDATA		#IMPLIED
 <b>labelNotes</b>	ANY			
▼ <b>labelNotes attribute list</b>				

### 7.7.4.Erläuterungen und Zuordnung zu GUIDO

#### <text>:EMPTY

positioniert an der jeweiligen Stelle des Auftretens den unter „content“ angegebenen Text. Optional kann im Attribut „position“ festgelegt werden, wie groß der Abstand des Textes vom Notensystem ist. Wird nichts angegeben, gilt ein Standardwert. <t> ist die Übersetzung der GUIDO-Kurzform \t, die Handhabung ist identisch.

*Beispiel:*

```

<note name="a" octave="0" duration="1/4"/>
<text content="Figaro, figaro" position="-4"/>
<note name="g" duration="1/4" accidentals="#" />
<note name="c" duration="1"/>

```

*in GUIDO:* a0/2 \text<"Figaro, Figaro",-4> g# c1

**<comment>:EMPTY**

fügt an der betreffenden Stelle einen Kommentar ein. Der eigentliche Inhalt des Kommentars steckt im Attribut „content“.

*Beispiel:*      <comment content="Ich bin ein Kommentar!"/>

*in GUIDO:*      (\* Ich bin ein Kommentar! \*)

**<title>:EMPTY**

dient zur Angabe des Titels des Musikstückes. Der eigentliche Name steht im Attribut „name“.

**<composer>:EMPTY**

beinhaltet den Komponisten des Musikstückes. Der eigentliche Name steht im Attribut „name“.

**<mark>:EMPTY**

kennzeichnet die entsprechende Stelle, an der der Tag auftritt mit dem im Attribut „name“ gespeicherten Wert.

*Beispiel:*      <mark name="ch:c7">  
                 <chord>  
                     <note name="c" duration="1/4"/>  
                     <note name="c" accidentals="b"/>  
                     <note name="g" duration="1/4"/>  
                 </chord>

*in GUIDO:*      \mark<"ch:c7"> {c1/4,e&,g}

**<label>?EMPTY**

verhält sich sehr ähnlich zu <mark>. Im Unterschied dazu lassen sich mit <label> ganze Stücke oder fest definierte Bereiche beschriften. Der Tag ist also sowohl als EMPTY als auch in der Form <labelNotes> als zweigeteilter Tag zu verwenden.

*Beispiel:*      <note name="c" duration="1/2"/>  
                 <note name="g" duration="1/4"/>  
                 <labelNotes name="Motiv1">  
                     <note name="g" duration="1/8"/>  
                     <note name="f" duration="1/8"/>  
                     <note name="e" duration="1/2"/>  
                     <note name="f" duration="1/4"/>  
                     <note name="d" duration="1/2"/>  
                 </labelNotes>

*in GUIDO:*      c1/2 g/4 \label<"a1">(g1/8 f e/2 f/4 d/2)

### 7.7.5.Überblick – Teil3

Elm staff	EMPTY			
▲ staff attribute list				
	Att Name	Att Type	Att Values	Att Presence
1	number	CDATA		#IMPLIED
2	name	CDATA		#IMPLIED
Comment	Beams			
Elm beamsAuto	EMPTY			
Elm beamsOff	EMPTY			
Elm beam	ANY			
▲ beam attribute list				
	Att Name	Att Type	Att Values	Att Presence
1	distance1	CDATA		#IMPLIED
2	distance2	CDATA		#IMPLIED
Elm bm	ANY			
▼ bm attribute list				
Comment	Stems			
Elm stemsAuto	EMPTY			
Elm stemsUp	EMPTY			
Elm stemsDown	EMPTY			
Comment	Bars			
Elm b	EMPTY			
▼ b attribute list				
Elm bar	EMPTY			
▲ bar attribute list				
	Att Name	Att Type	Att Values	Att Presence
1	measure	CDATA		#IMPLIED
Elm doubleBar	EMPTY			
▲ doubleBar attribute list				
	Att Name	Att Type	Att Values	Att Presence
1	measure	CDATA		#IMPLIED
Elm tactus	EMPTY			

### 7.7.6.Erläuterungen und Zuordnung zu GUIDO

#### <staff>:EMPTY

setzt die Stimme, an deren Anfang der Tag steht in das Notensystem. Die Bezeichnung des Systems kann wahlweise über eine Zahl (number:"1") oder einen Namen (name:"Sopran") erfolgen.

*Beispiel:*

```

<segment>
  <sequence>
    <staff number="1"/>
      <note name="e" duration="1/8"/>
      <note name="d" duration="1/8"/>
      <note name="c" duration="1/8"/>
    </sequence>
  <sequence>
    <staff number="1">
      <note name="c" duration="1/8" dot="yes"/>
  
```

```

    </sequence>
  </segment>

```

in GUIDO: { [\staff<1> e1/8 d c], [\staff<1> c1/8.] }

## <beam...>?EMPTY

legt fest, wie Noten gleicher Länge miteinander durch Balken verbunden werden. Mit den EMPTY-Tags **<beamsAuto>** und **<beamsOff>** kann die Balkensetzung automatisch erfolgen bzw. völlig unterbunden werden. Die zwei optionalen Attribute „distance1“ und „distance2“ können verwendet werden, um die Abstände des Balkens von den Noten festzulegen. **<bm>** entspricht der GUIDO-Kurzform **\bm** und hat die identische Funktionalität.

Beispiel:

```

<beam>
  <note name="d" duration="1/16"/>
  <note name="e" duration="1/16"/>
  <note name="c" duration="1/8"/>
</beam>
<beamsOff/>
  <note name="d" duration="1/8"/>
  <note name="e" duration="1/8"/>

```

in GUIDO: \beam(d1/16 e c/8) \beamsOff d e

## <stems...>:EMPTY

nimmt Einfluss auf die Ausrichtung der Notenhäse. Mit **<stemsUp>** und **<stemsDown>** kann für die auf die Tags folgenden Noten erzwungen werden, dass die Häse nach oben bzw. nach unten zeigen. **<stemsAuto>** wechselt zurück in den Standardmodus, in dem die Ausrichtung automatisch erfolgt.

Beispiel:

```

<stemsUp/>
  <note name="c" duration="1/4"/>
  <note name="e" duration="1/4"/>

```

in GUIDO: \stemsUp c1/4 e

## <bar>:EMPTY

## <doubleBar>:EMPTY

## <tactus>:EMPTY

Mit **<bar>** und **<doubleBar>** können manuelle Takt- und Doppelstriche an bestimmten Stellen innerhalb des Musikstücks eingefügt werden. Das Attribut „measure“ gibt darüber hinaus die Zahl des Taktes an, der auf den Taktstrich folgt. **<tactus>** kennzeichnet rhythmische Unterteilungen mit gepunkteten Linien oder Kommas.

Beispiel:

```

<note name="c" duration="1/2"/>
<note name="e" duration="1/4"/>

```

```
<note name="e" duration="1/4"/>  
<doubleBar/>  
<rest duration="1/4"/>  
<note name="f" duration="1/4"/>  
<note name="a" duration="1/2" accidentals="b"/>
```

*in GUIDO:*    c1/2 e/4 e \doubleBar \_ f a&/2

## **8.Implementierung GUIDO2XML**

Das Programm zur Konvertierung von Dateien in der GUIDO-Notationssprache in die definierte XML-konforme Sprache ist vollständig in ANSI C geschrieben und ist daher unter allen gängigen Betriebssystemen lauffähig.

Die Aufbereitung des eingelesenen Quelltextes der GUIDO-Datei in eine Liste von Tokens wird vom GUIDO-Parserkit in der Version 0.61 erledigt, der am Fachgebiet entwickelt worden ist.

Das GUIDO-Parser-Kit verwendet Callback-Funktionen. Die Implementierung dieser Funktionen, ist Hauptbestandteil des guido2xml-Konverters.

Darüber hinaus gibt es eine Tag-Liste, in der die entsprechenden GUIDO-XML-Tags zusammengefasst sind. Hier kann der Sprachumfang auch erweitert oder geändert werden. Nähere Informationen und Kommentierung sind im Quelltext verfügbar.

Unser Vorgehen richtete sich nach folgende Gliederung:

- Konzept entwickeln
- Recherche
- Technische Realisation
- Dokumentation

### **8.1.Entwicklung des Konzepts**

Hier ging es hauptsächlich um die Fragen:

- Welche Programmiersprache soll zugrunde gelegt werden?
- Wie soll der Parser funktionieren?
- Welche Bedienungsmöglichkeit soll das Programm haben?
- In wie weit soll Robustheit und Erweiterbarkeit eine Rolle spielen?

Die Frage nach einem geeigneten Parser war schnell entschieden, da durch das GUIDO-Parser-Kit v0.61 alle nötigen Features zu Verfügung standen. Deshalb wurde die Programmiersprache C/C++ gewählt, da das Parser-Kit ebenfalls in dieser Programmiersprache geschrieben wurde. Außerdem ist C/C++ ein recht weit verbreiteter Standard und auf fast allen Computersystemen verfügbar. Beim Programmieren wurde darauf geachtet dem ANSI-Standard so weit wie möglich gerecht zu werden.

Das GUIDO-Parser-Kit verwendet beim Parsen Callback-Funktionen. Diese sind fest deklariert und müssen nur noch implementiert werden. Die Callback-Funktionen werden beim Parsen aufgerufen, falls GUIDO-konforme Tags gefunden werden. D.h. das GUIDO2XML-Programm wird zur Laufzeit des Parsens sozusagen vom Parser getriggert. Aus diesem Grund ist es sinnvoll ein Filestreaming zu benutzen, um nach jedem Triggerschritt die entsprechenden Daten in die Ausgabedatei zu schreiben. Das vermindert die Störanfälligkeit des Konverters. Außerdem wird ein dynamischer Stack benutzt, mehr dazu später.

Um eine gewisse Erweiterbarkeit zu gewährleisten legten wir besonderen Wert auf die folgenden Punkte:

1. Globale Definitionen sollen möglichst leicht und übersichtlich editierbar sein.
2. Tagänderungen, bzw. Tagerweiterungen sollen ebenfalls leicht zugänglich sein.
3. Übersichtliche und aussagekräftige Quellcodedokumentation.
4. Die Codebase soll übersichtlich gestaltet sein.

Die Bedienung des Programms soll möglichst funktional sein, d.h. es gibt eine Eingabe in Form eines Parameters und eine Ausgabe in Form einer Datei. Warnungen bzw. Fehler werden entsprechend ausgegeben.

## **8.2.Recherche**

Die Recherche diene zum größten Teil der Informationsbeschaffung. Hier sollte geklärt werden:

- Welcher Parser kann benutzt werden?
- Welche zusätzlichen Programme sind evtl. notwendig?
- Sind Quellcodes von anderen Projekten notwendig?
- Welche Lizenzen sind in anderen Projekten vorhanden?

Die Recherche hat ergeben, dass das GUIDO-Parser-Kit alle nötigen Programm-Module mit sich bringt um, in der Praktikumszeit das entsprechende Programm fertig zu stellen. Des weiteren wurde Wert auf UNIX- und Win32-Unterstützung gelegt. Für UNIX ist lediglich ein GNU-C/C++ Compiler mit den entsprechenden Utilities (make, ...) notwendig. Die Win32-Version baut auf Visual C/C++ von Microsoft auf (siehe hierzu die *Readme.txt*).

## **8.3.Technische Realisation**

Dieser Abschnitt war zeitlich gesehen der größte Aufwand, da man hier erst erkennen konnte, welche Merkmale überhaupt Sinn machen, bzw. geändert werden müssen. Ein nicht zu verachtender Teil wurde für die nachträgliche Fehlerkorrektur verwendet. Zunächst wurde so schnell wie möglich eine lauffähige Vorabversion erstellt, um sich mit dem Parser-Kit vertraut zu machen. Hier war dann auch zu erkennen, wo es Probleme geben kann (z.B. werden geklammerte Ausdrücke nicht explizit angegeben).

### **8.3.1.Verzeichnisstruktur**

Die Verzeichnisstruktur wurde so gewählt, dass man einen übersichtlichen Eindruck gewinnt:

guido2xml	
- bin	(für ausführbare Dateien)
- dtd	(der eigentliche DTD)
- examples	(Beispieldateien)
- guido-parser-kit-0.61	(das GUIDO-Parser-Kit)
- obj	(erstellte Objektdateien)
- src	(der Quellbaum)
- win32	(Visual C/C++ Dateien für Windows)
Readme.txt	(Kurzbeschreibung)
Makefile	(Makefile für UNIX)

Die Datei defines.h definiert alle globalen Konstanten. Hier steht z.B. die GUIDO2XML-Version oder die Tabulatorlänge des Pretty-Prints, etc.



Das File- (file.h, file.cpp), Stack- (stack.h, stack.cpp) und Taghandling (taglist.h, taglist.cpp) wurde ausgelagert.

### 8.3.2.Filehandling

Das Filehandling implementiert die File-Stream Operationen, wie z.B. öffnen, schließen und schreiben.

### 8.3.3.Stackhandling

Da geklammerte Tags vorkommen können wurde, ein Stack implementiert, um einem Tag-Begin das richtige Tag-End zuordnen zu können. Das Stackhandling implementiert einen dynamischen Stack, d.h. es wird entsprechend neuen Speicher angefordert, falls Daten auf den Stack geschrieben werden. Andernfalls wird natürlich der Speicher wieder freigegeben.

### 8.3.4.Taghandling

Das Taghandling besitzt alle Funktionen zur Tagverwaltung. Als Grundlage werden die Tags in Stringlisten gespeichert. Eine Stringliste besteht immer aus GUIDO-Tag, XML-Tag und den Tag-Parametern. Es gibt Funktionen, welche die Tags, bzw. Parameter finden können. Näheres dazu findet man in der Datei taglist.h.

Es gibt drei Grundklassen von Taglisten:

1. Geklammerte Tags (bracketed)
2. Mehrdeutige Tags (ambiguous)
3. Nicht geklammerte Tags (bracketed\_none)

**Wichtig:** Falls kein Tagname oder Tag-Parametername gefunden wurde, wird versucht den entsprechenden (falls in der Tagliste vorhanden) zu ersetzen. Falls kein Tag-Parametername vorhanden ist, wird als Standardname **argN** benutzt (N ist aus 0...n).

### 8.3.5.Mehrdeutige Tags

Mehrdeutige Tags können sowohl geklammert als auch nicht geklammert vorkommen. Als Entscheidung für einen nicht geklammerten Tag wird überprüft, ob der Tag leer ist, d.h. Tag-Begin und Tag-End erfolgt unmittelbar aufeinander.

**Wichtig:** Die mehrdeutigen Tags sollten jedoch ausgespart bleiben, da die DTD in XML keine adäquate Definition zulässt (man müsste die Tags als **Empty** und **Any** deklarieren, was jedoch ein Widerspruch darstellt).

### 8.3.6.Typabhängige Tags

Es ist zudem möglich, Tag-Parameter abhängig vom Typ (int, float, string, unit) zu deklarieren. Die Priorität dieser Liste ist höher als die der vorangenannten, d.h. falls typabhängige Tags gefunden wurden, werden nur die Daten dieser Tagliste geltend gemacht (natürlich nur falls Daten vorhanden sind). Zu jedem Typ gibt es eine Stringliste.

**Wichtig:** Der Taglisten-Verwaltung (Tag-Operationen) wurde keine semantische Bedeutung zugeordnet. Dies erfolgt in *guido2xml.cpp*. Damit ist die Taglistenverwaltung generisch und hat keinerlei Einschränkung hinsichtlich der Erweiterbarkeit.

### **8.3.7.Tag-Rule-Liste**

Als letztes sei noch die Tag-Rule-Liste erwähnt. Hier können Tag-Parameter abhängig von der Anzahl der tatsächlichen Parameter definiert werden. Dazu müssen alle Parameter in gewünschter Reihenfolge vorliegen. Als Parameter gibt man nun die Split-Points an. Diese geben nur an, ab welcher Stelle die Parameterliste abgearbeitet werden soll. Die Parameter werden immer von 1 bis n durchnummeriert. Es wird immer das Maximum der verfügbaren Split-Points gesucht. Dabei darf der gefundene Split-Point nicht größer als die Anzahl der übergebenen Parameter sein. Man braucht also nur noch die Stellen angeben, die die Liste richtig (in kleinere Listen) aufteilt. Näheres hierzu in *taglist.h*.

### **8.4.Dokumentation**

Der letzte Teil der Arbeit beschäftigte sich mit der Dokumentation des Quelltextes. Dabei wurde darauf geachtet alle Funktionen kurz zu beschreiben und kompliziertere Programmfragmente zu kommentieren.

## **9.Implementierung XML2GUIDO**

Guido2XML ist ein kommandozeilen-orientiertes Konvertierungstool, das Dateien im Guido-XML-Format (wie in diesem Dokument beschrieben) in das GUIDO-Format übersetzt. Es ist komplett in C gehalten und jederzeit erweiterbar. Das Programm basiert im wesentlichen auf dem XML-Parser Expat, Ver. 1.95.1, von James Clark, der unter anderem Technischer Leiter bei der XML Working Group des W3-Konsortiums war, die die XML-Spezifikationen entwickelt haben.

### **9.1.Programmbenutzung**

Die Benutzung des Programms ist sehr einfach. Auf der Kommandozeilenebene gibt man den Befehl "guido2xml datei.xml" ein, wobei "datei.xml" der Dateiname der zu übersetzenden XML-Datei ist. Die XML-Datei wird in eine gleichnamige GMN-Datei (Guido-Format) übersetzt, gleichzeitig wird eine Datei "datei.uts" erzeugt, die eventuelle unbekannte Tags und ihr Vorkommen in der XML-Datei auflistet.

### **9.2.Vorgehensweise des Programms**

Guido2XML greift während der Übersetzung hauptsächlich auf zwei von Expat zur Verfügung gestellte Routinen zu, start und end. Nachdem die XML-Datei geöffnet und einem Parser-Objekt p übergeben wurde, wird bei jedem öffnenden XML-Tag der Form **<tag>** die Routine **start**, bei jedem schließendem XML-Tag der Form **</tag>** die Routine **end** aufgerufen. Bei XML-Tags der Form <tag/>, einem Single-Tag also, werden die beiden Routinen direkt hintereinander gestartet.

Alle Start- und Endtags vergleicht Guido2XML mit einer Tagliste, die in der Datei "taglist.h" definiert wird. Die Tagliste hat das Format:

```
xml-tag:behavior:guido-tag:param1:param2:...:paramX.
```

xml-tag ist der zu vergleichende XML-Tagname, guido-tag sein Äquivalent im Guido-Format, behavior gibt an, ob es sich um einen ungeklammerte(u), geklammerten(b) oder variablen(a) Tag handelt, aus dieser Angabe ergibt sich die Parse-Art des behandelten Tags. Die Einträge param1 bis paramX geben die Parameter an, welche von Guido2XML ausmaskiert werden sollen. Die Werte aller hier angegebenen Parameter werden also direkt in die GMN-Datei geschrieben. Sollten in dem behandelten Tag Parameter vorkommen, die hier nicht gelistet sind, so wird der Parametername und sein Wert mit einem Gleichheitszeichen in die GMN-Datei übernommen.

#### **Beispiel:**

Listeneintrag in taglist.h:      staff:u:staff:i:str

XML-Datei:                      <staff i="1" dy="0.850000cm"/>

GMN-Datei:                      \staff<1,dy=0.850000cm>

Der Wert von i als gelistetem Parameter wurde direkt geschrieben, der ungelistete Parameter dy wurde mit Namen übernommen.

Ist ein behandelter XML-Tag nicht in der Tagliste aufgeführt, so parst Guido2XML den Tag in einem Standardverfahren. Daraus resultiert dann folgende Darstellung:

#### XML-Datei:

```
<fingering text="&#x22;3&#x22;" dy="8hs" dx="-0.100000hs" fsize="7pt">  
  <note name="f" octave="2" duration="1/16" accidentals="#" />  
</fingering>
```

#### Guido-Datei:

```
\fingering<text="3",dy=8hs,dx=-0.100000hs,fsize=7pt>f#2*1/16
```

Der unbekannte Tag wird also einfach in die Guido-Datei übernommen, seine Parameter werden wie unbekannte Parameter behandelt. Der Name des unbekannten Tags sowie sein Vorkommen in der XML-Datei werden in die Datei **"datei.uts"** geschrieben.

### **9.3.Erweiterbarkeit**

Guido2XML ist sehr einfach zu erweitern. Neue Tags müssen dazu nach obigem Muster in die Tagliste "taglist.h" hinzugefügt werden, in der Datei selber steht eine detailliertere Anleitung dazu. Anschließend muß das System neu kompiliert werden, am besten mit einem "make clean" unter Linux oder einem "rebuild all" unter Visual C++.

## **10.Literaturverzeichnis**

- [EC00] Robert Eckstein, „XML kurz & gut“, O'Reilly Taschenbibliothek, Juli 2000
- [KO99] Thomas Kobert, „XML“, bhv Taschenbuch, 1999
- [HA98] Elliotte R. Harold, „XML Extensible Markup Language“, IDG Books Worldwide, 1998