Goobi Content Server

Installation und Anwendung

Goobi Content Server: Installation und Anwendung

Steffen Hankiewicz, Markus Enders, Igor Toker

Zusammenfassung

Dieses Whitepaper dient als kurzes Übersichtsdokument für die Installation und den Betrieb des Goobi Content Servers. Die Softwareentwicklung fand als Zusammenarbeit zwischen dem Göttinger Digitalisierungszentrum und intranda software statt und liegt als Open Source zur freien Verwendung unter der GNU General Public License vor.

Inhaltsverzeichnis

1. Über dieses Dokument

Dieses Dokument versteht sich als Kurzanleitung für die Installation und den Betrieb des Goobi Content Servers. Da die Software weiterentwickelt wird, unterliegt auch dieses Dokument einer gelegentlichen Überarbeitung. An dieser Stelle folgt daher eine Übersicht über die einzelnen Revisionen dieses Dokuments unter Angabe der aktuellen Revisionsnummer sowie der Version des Goobi Content Servers, worauf diese Dokumentation sich bezieht.

Bitte beachten Sie, dass die Revision dieser Dokumentation zu Ihrer installierten Version des Goobi Content Servers passt, um Fehler zu vermeiden. Eine aktuelle Fassung sowohl des Goobi Content Servers als auch dieses Whitepapers finden Sie auf den Webseiten des Göttinger Digitalisierungszentrums http://gdz.sub.uni-goettingen.de sowie auf den Webseiten von intranda software unter http://www.intranda.com

Revision	Datum	Version	Änderung
0.1	02.04.2009	0.8	Erste Zusammenstellung des Inhalts
0.2	08.04.2009	0.9	Allgemeine Erläuterungen zum Dokument und Projekt
0.3	10.04.2009	1.0	Erläuterungen zu den Java-Projekten
0.4	15.04.2009	1.0	Beschreibung zu und Konfiguration der Wasserzeichen
0.5	19.04.2009	1.0	Beschreibung und Konfiguration der PDF-Titelseiten
0.6	23.04.2009	1.0	Installation des ContentServers
0.7	25.04.2009	1.0	URL-Parameterbeschreibung
0.8	26.04.2009	1.0	API 1
0.9	06.05.2009	1.0	API 2
0.10	13.05.2009	1.0	Dokumentation zu den action Befehlen ergänzt

Historie der Revisionen

2. Allgemeines

2.1 Aufgaben des Goobi Content Servers

Bei dem Goobi Content Server handelt es sich um eine Open Source Library zur automatischen Verarbeitung verschiedener Image- und PDF-Operationen. Ihre Anwendung findet hauptsächlich im Kontext der Digitalisierung von Büchern insbesondere für die Präsentation von Digitalisaten statt. Dafür bietet der Goobi Content Server verschiedene Methoden, um die Images zu manipulieren und in andere Bildformate zu übertragen. Auch bieten sich verschiedene Möglichkeiten, aus Images PDF-Dateien zu generieren oder auf der Basis von METS-Dateien mit enthaltenen Struktur und Metadaten unter Angabe der relevanten Dateigruppen digitalisierte Werke als PDF-Dokumente inkl. Inhaltsverzeichnissen und Watermarks zu erzeugen.

2.2 Die verschiedenen Einzelprojekte des Goobi Content Servers

Die gesamte Library des Goobi Content Servers besteht aus insgesamt mehreren einzelnen JAVA-Projekten:

- ContentServer: Manipulation von Images (Skalierung, Rotation, Highlighting, Watermarks, Dateiformate etc.) sowie Erzeugung von PDF-Dateien aus einer Liste von Images mit der Möglichkeit zur Erzeugung von Titelseiten sowie Inhaltsverzeichnissen; sämtliche dieser Funktionen stellt der ContentServer ausserdem über eine Weboberfläche zur Verfügung
- SimpleMets: Verarbeitung von Mets-Dateien zur Ermittlung von Image-Dateien bestimmter Dateigruppen, Verarbeitung von Struktur- und Metadaten und Generierung von PDF-Dokumenten auf Basis der METS-Dateien
- PDF-Service: Erzeugung von PDF-Dateien auf der Basis mehrerer METS-Dateien sowie mehrerer PDF-Dateien; die Inhalte werden in einem großen Dokument inklusive Inhaltsverzeichnis zusammengeführt
- Goobi Content Server Servlet: Bereitstellung sämtlicher Möglichkeiten der Einzelprojekte des Goobi Content Servers über eine Weboberfläche

Einzelne Projekte des Goobi Content Servers

3. Installation und Konfiguration

An dieser Stelle sollen einige Hinweise zur Installation der beiden Servlets zusammengeführt werden. Auf eine detaillierte Beschreibung der Installation wird dabei verzichtet. Es wird davon ausgegangen, dass ein Rechner bereits über eine Java Virtuell Machine sowie einen installierten Servlet Container (z.B. Apache Tomcat) verfügt.

Bitte beachten Sie, dass für den korrekten Betrieb des Goobi Content Servers eine Java Laufzeitumgebung der Version 5 oder neuer vorliegen muss. Unter älteren Javaversionen ist ein Betrieb des Goobi Content Severs nicht möglich.

3.1 Installation der Servlets

Ein Servlet wird wie folgt (Beispiel Tomcat) installiert:

1. Die .war-Datei herunterladen
2. Die .war-Datei in den Servlet-Container (im Tomcat ist es das /webapps Verzeichnis) schieben/bzw. kopieren

Servlet Installation

Da es sich bei dem Servlet um Java 5 Quellcode handelt muss ggf. innerhalb des Tomcats noch eine Änderung an der Datei *TOMCAT_INSTALL\conf\web.xml* erfolgen, die auch unter http://firstclassthoughts.co.uk/java/tomcat_5_eclipse_source_1.5.html erläutert wird:

```
<init-param>
   <param-name>compilerSourceVM</param-name>
   <param-value>1.5</param-value>
</init-param>
<init-param>
   <param-name>compilerTargetVM</param-name>
   <param-value>1.5</param-value>
</init-param>
```

Anpassungen Tomcat für Java5

Weiterführende Informationen können unter http://www.apache.org gefunden werden.

3.2 Konfiguration des Content Servers

Hier ein Beispiel für die Konfiguration des Content Servers

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

 <!--#########################################################
    *#########################################################
    *
    *                  general parameters
    *
    *#########################################################
    *#########################################################-->

 <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * default resulution for results of image processing
    * sample: <defaultResolution value="600" />
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <defaultResolution value="600" />

    <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * default file names for pdf or image (file extension will be automatically added);
    * define if sending of images or pdf should be with header "attachment" or not;
    * currently available variables for file name:
    *   - $datetime         current date and time (pdf and images)
    *
    * sample:
    * <defaultFileNames>
    *    <pdf value="GoobiContentServer_$datetime" sendAsAttachment="true"/>
    * </defaultFileNames>
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <defaultFileNames>
        <pdf value="ContentServer_$datetime" sendAsAttachment="true"/>
        <image value="image_$datetime" sendAsAttachment="false"/>
    </defaultFileNames>

    <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * configuration for the pdf generation, these parameters will change the performance of pdf generation
    *   - alwaysUseRenderedImage           convert all images always to rendered image before sending it to iText
    *   - alwaysCompressToJPEG             compress all images always to jpeg image before sending it to iText
    *   - metsFileGroup                    the filegroup to use for pdf generation
    *   - writeAsPdfA                      write PDF file as Pdf/A
* - pagesize         default page size for pdf generation; possible values are 'A4', 'original' and 'A4Box'
    * sample: <defaultPdfConfig alwaysUseRenderedImage="true" alwaysCompressToJPEG="true" metsFileGroup="DEFAULT"
writeAsPdfA="true" pagesize="A4"/>
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <defaultPdfConfig alwaysUseRenderedImage="true" alwaysCompressToJPEG="true" metsFileGroup="DEFAULT" writeAsPdfA="true"
pagesize="A4"/>

    <!--#########################################################
    *#########################################################
    *
    *                  ContentServer parameters
    *
    *#########################################################
    *#########################################################-->

    <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* default path to repository for all image processing requests
* if repository path is empty, in the servlet request has to be a complete url
* sample: <defaultRepositoryPath value="file:/home/goobi/imageRepository/" />
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<defaultRepositoryPathImages
 value="file:/home/goobi/imageRepository/" />

    <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * default color for highlighting inside the image as RGB
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <defaultHighlightColor valueRed="255" valueGreen="255"
 valueBlue="0" valueAlpha="100"/>

    <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * configuration if watermark should be used and url for watermark content configuration file
    * sample: <watermark use="true" configFile="file:/home/goobi/config/watermarkconfig.xml" />
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<watermark use="false" configFile="file:/home/goobi/config/watermarkconfig.xml" />
```

Konfiguration des Content Server

```
   <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   * configuration for the error watermark if message should not be shown on jsp echo page
   *   - title                title on top of image
   *   - titleSize            fontsize for title
   *   - messageSiz           fontsize for error message
   *   - messageLineLength    maximum size of letters for each line
   *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
 <errorWaterMark title="Error" titleFontSize="20" messageFontSize="14" messageMaxLineLength="60"/>
</config>
```

Konfiguration des Content Server

3.3 Konfiguration des Goobi Content Servers

Hier ein Beispiel für die Konfiguration des Goobi Content Servers. Im Prinzip wie oben aber mit ein paar zusätzlichen Einstellungen

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

  <!--##########################################################
    *##########################################################
    *
    *                 general parameters
    *
    *##########################################################
    *##########################################################-->

 <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * default resulution for results of image processing
    * sample: <defaultResolution value="600" />
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <defaultResolution value="600" />

    <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * default file names for pdf or image (file extension will be automatically added);
    * define if sending of images or pdf should be with header "attachment" or not;
    * currently available variables for file name:
    *   - $datetime         current date and time (pdf and images)
    *
    * sample:
    * <defaultFileNames>
    *    <pdf value="GoobiContentServer_$datetime" sendAsAttachment="true"/>
    * </defaultFileNames>
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <defaultFileNames>
        <pdf value="ContentServer_$datetime" sendAsAttachment="true"/>
        <image value="image_$datetime" sendAsAttachment="false"/>
    </defaultFileNames>

    <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * configuration for the pdf generation, these parameters will change the performance of pdf generation
    *   - alwaysUseRenderedImage           convert all images always to rendered image before sending it to iText
    *   - alwaysCompressToJPEG             compress all images always to jpeg image before sending it to iText
    *   - metsFileGroup                    the filegroup to use for pdf generation
    *   - writeAsPdfA                      write PDF file as Pdf/A
 * - pagesize         default page size for pdf generation; possible values are 'A4', 'original' and 'A4Box'
    * sample: <defaultPdfConfig alwaysUseRenderedImage="true" alwaysCompressToJPEG="true" metsFileGroup="DEFAULT"
 writeAsPdfA="true" pagesize="A4"/>
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <defaultPdfConfig alwaysUseRenderedImage="true" alwaysCompressToJPEG="true" metsFileGroup="LOCAL" writeAsPdfA="true"
 pagesize="A4"/>
```

Konfigurations des Goobi Content Server

```
   <!--#########################################################
   *#########################################################
   *
   *                  ContentServer parameters
   *
   *#########################################################
   *#########################################################-->

   <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * default path to repository for all image processing requests
 * if repository path is empty, in the servlet request has to be a complete url
 * sample: <defaultRepositoryPath value="file:/home/goobi/imageRepository/" />
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
 <defaultRepositoryPathImages
  value="file:/home/goobi/imageRepository/" />

   <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   * default color for highlighting inside the image as RGB
   *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
   <defaultHighlightColor valueRed="255" valueGreen="255"
  valueBlue="0" valueAlpha="100"/>

   <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   * configuration if watermark should be used and url for watermark content configuration file
   * sample: <watermark use="true" configFile="file:/home/goobi/config/watermarkconfig.xml" />
   *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
 <watermark use="false" configFile="file:/home/goobi/config/watermarkconfig.xml" />

   <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   * configuration for the error watermark if message should not be shown on jsp echo page
   *   - title                title on top of image
   *   - titleSize            fontsize for title
   *   - messageSiz           fontsize for error message
   *   - messageLineLength    maximum size of letters for each line
   *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
 <errorWaterMark title="Error" titleFontSize="20" messageFontSize="14" messageMaxLineLength="60"/>

   <!--#########################################################
   *#########################################################
   *
   *                  GoobiContentServer parameters
   *
   *#########################################################
   *#########################################################-->

  <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * default path to repository for all mets processing requests
 * if repository path is empty, in the servlet request has to be a complete url
 * sample: <defaultRepositoryPath value="file:/home/goobi/metsRepository/" />
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
 <defaultRepositoryPathMets
  value="file:/home/goobi/metsRepository/" />

   <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * configuration if titlepage for pdf file should be generated and
    * which config-file defines how the title page should look like
    *
    * sample: <defaultRepositoryPath value="file:/home/goobi/config/pdftitlepage.xml" />
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <pdfTitlePage use="false"
        configFile="file:/home/goobi/config/pdftitlepage.xml/pdftitlepage.xml" />

     <!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    * path to contentCache and maximum size in MB
    *   -path: path in file system
    *   -size: size in MB
    *   -useCache: central switch, if no cache at all should be used
    * -useShortFileNames: define if the cached file name should only consist of given url-parameter and divid
 (usShortFileNames=true, not recommended)
    *   or if it should be full named (useShortFileNames=false, more secure option)
    *
    * sample: <contentCache useCache="false" path="/home/goobi/gcsCache" size="300"/>
    *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
    <contentCache useCache="true" path="/home/goobi/gcsCache" size="30" useShortFileNames="false"/>
</config>
```

Konfigurations des Goobi Content Server

3.4 Wasserzeichen

Generell ist ein digitales Wasserzeichen (Watermark) eine Information, die in die Trägerinformation eingefügt wird. Im Kontext des Contentservers dient das Wasserzeichen dazu, Herkunfts- oder Copyrightinformation zu transportieren. Diese Information wird in ein Image eingefügt, welches als Trägermedium dient. Im Gegensatz zur Steganographie handelt es sich hierbei um sichtbare Wasserzeichen. Das Wasserzeichen ist in dem Trägerimage jederzeit sichtbar.

Anwendungszweck eines solchen Wasserzeichens ist es, Copyright-Information im Image unterzubringen. Damit das eigentliche Image jedoch in allen Fällen noch lesbar bleibt, kann ein Wasserzeichen nur an den Rand (oben, unten, rechts, links) des Trägerimages angefügt werden. Das Anfügen von Wasserzeichen in PDF-Dokumenten wird nicht unterstützt.

Intern hat ein Wasserzeichen immer eine feste Grösse. Diese ist je nach Ausrichtung immer mit der Höhe oder Breite des Trägerimages identisch.

Beispiel eines generierten Wasserzeichens

3.4.1 Komponenten eines Wasserzeichens

Ein Wasserzeichen besteht aus einer oder mehrerer Komponenten. Diese werden zur Laufzeit gerendert, wenn das Wasserzeichen gerendert werden soll. Änderungen an den Komponenten bzw. derer Inhalt zur Laufzeit werden daher sofort berücksichtigt. Es muss kein neues Watermark instantiiert werden. Folgende Komponententypen werden unterstützt:

- *Box*: eine gefüllte Box
- *Text*: ein Text in beliebiger Größe

- *Image*: ein Image, welches in einem der durch den ContentServer unterstützten Formate vorliegen muss.

Komponententypen für Wasserzeichen

Diese Komponenten werden in einer Konfigurationsdatei definiert. Die Reihenfolge der Definition bestimmt die Reihenfolge, wie die Komponenten gerendert werden. Die erste Komponente wird in den Hintergrund gerendert und kann ggf. durch nachfolgend definierte Komponenten überschrieben werden.

Jede Komponente verfügt über eine Koordinatenangabe. Diese gibt die Position der Komponente relativ zum Nullpunkt des Wasserzeichens an und berücksichtigt die horizontale Ausrichtung der Komponente. Jede Komponente kann entweder linksbündig, rechtsbündig oder zentriert ausgerichtet werden. Eine vertikale Ausrichtung ist nicht möglich.

- *Linksbündig*: bei linksbündiger Ausrichtung befindet sich der Nullpunkt an der oberen linken Ecke des Wasserzeichens. Die Koordinaten geben ebenfalls den oberen linken Punkt der Komponente an.
- *Rechtsbündig*: bei rechtsbündiger Ausrichtung befindet sich der Nullpunkt an der oberen rechten Ecke des Wasserzeichens. Die Koordinaten geben ebenfalls den oberen rechten Punkt der Koordinate an.
- *Zentriert*: der Wert der x-Koordinate wird nicht berücksichtigt.

Ausrichtung der Wasserzeichenkomponenten

3.4.2 Generierung eines Wasserzeichens

Bei der Bildkonvertierung kann auf Wunsch ein Wasserzeichen automatisch mit generiert werden. Die entsprechende Einstellung wird in der URL zur Bildkonvertierung angegeben. Neben dem Pfad zur Konfigurationsdatei müssen auch Positions- und Skalierungsinformationen angegeben werden.

Die Positionierungsinformation bestimmt, an welcher Seite das Wasserzeichen zum Image angefügt wird. Die Skalierungsinformation bestimmt, ob das Wasserzeichen skaliert oder gestreckt werden soll, um es entsprechend der Höhe oder Breite des Images anzupassen.

Wird das Wasserzeichen skaliert, so wird die Breite/Höhe proportional zum Image geändert. D.h. die in der Konfigurationsdatei angegebene Breite/Höhe des Wasserzeichens dient als Grundlage zur Bestimmung eines Skalierungsfaktors. Die Skalierung geschieht allerdings erst, nachdem das Wasserzeichen gerendert wurde. Es wird also das Rasterimage, welches über die in der Konfigurationsdatei angegebene Breite verfügt, skaliert. Bei Vergrößerung folgt daraus ein Qualitätsverlust.

Das Stecken des Wasserzeichens bedeutet, dass alle Komponenten entsprechend der neu ermittelten Wasserzeichenbreite/-höhe gerendert werden. Hierbei wird die Ausrichtung der Komponenten entsprechend berücksichtigt.

3.4.3 Konfiguration von Wasserzeichen

Wasserzeichen können entweder zur Laufzeit über die API oder aber über eine XML Datei konfiguriert werden. Die XML-Datei hat folgenden Aufbau:

Ein umschließendes <watermark> Element (= root-Element) definiert den Beginn des Wasserzeichens. Es kennt folgende Attribute:

- *width*: verpflichtendes Attribut, Angabe der Breite des Wasserzeichens in Pixel
- *height*: verpflichtendes Attribut, Angabe der Höhe des Wasserzeichens in Pixel
- *color*: Hintergrundfarbe des Wasserzeichens; wird gemäß HTML in 3 zusammengefassten hexadezimalen Werten angegeben.

watermark Attribute

Innerhalb des <watermark> Elements werden die Komponenten definiert. Die Anzahl der Komponenten ist unbegrenzt. Jede Komponente kennt die Attribute x und y, welche die Pixelkoordinaten der Komponente innerhalb des Wasserzeichens angeben. Diese Attribute sind verpflichtend. Wie oben erläutert hängen diese auch von der Ausrichtung der Komponente ab. Diese wird mittels des origin-Attributs festgelegt (optional). Das origin Attribut kennt folgende Werte (alle klein geschrieben):

- *left*: linksbündig ausgerichtet
- *right*: rechtbündig ausgerichtet
- *center*: zentriert ausgerichtet.

origin Attributwerte

Die Ausrichtung bezieht sich immer auf die Horizontale. Eine vertikale Ausrichtung ist nicht möglich. Hierbei bezieht sich der Wert des y-Attributs immer auf die obere Ecke.

Das Rechteck wird über das <box> Element definiert. Das Element ist leer; kennt also keine Unterelemente oder einen Wert. Als zusätzliches Attribut kann mittels des color-Attributs die Füllfarbe angegeben werden. Der Wert des color-Attributs ist die HTML-konforme Angabe für die Farbe.

Das <image> Element definiert ein Bild. Die URL des Bildes wird im url-Attribut angegeben. Dieses Attribut ist verpflichtend. Es werden alle Imagetypen des ContentServer unterstützt. Das Element selbst ist leer und darf keine Unterelemente oder Text enthalten.

Das <text> Element definiert einen Text. Der Text selbst ist Inhalt des Elements. D.h. das <text> Element darf nicht leer sein. Sollte der Text beim Rendern der Komponente

nicht in das Wasserzeichen passen, wird er einfach abgeschnitten. Der Text wird NICHT automatisch umbrochen. Das <text> Element kennt folgende Attribute:

- *fontname*: Name des Fonts; es werden zukünftig alle TrueType-Fonts, die die Java-Virtual-Machine kennt, unterstützt. Dieses Attribut ist verpflichtend.
- *fontsize*: Größe des Textes in Punkt. Dieses Attribut ist verpflichtend.
- *fontcolor*: Farbe des Textes gemäß HTML-Farbangabe (6 hexadezimale Werte). Dieses Attribut ist optional.

test Attribute

Die verwendeten TrueType Fonts sollten der JAVA-Virtual-Machine verfügbar sein. Ansonsten wird die Standardschriftart Arial genutzt.

Das nachfolgende Beispiel zeigt eine Konfigurationsdatei für ein Wasserzeichen:

```
<?xml version="1.0" encoding="UTF-8"?>
<watermark width="600" height="80" color="FFFFFF">
   <box x="30" y="50" height="65" width="180" color="555555"/>
   <image x="35" y="15" origin="left"
      url="http://intranda.com/logo1.png"/>
   <image x="35" y="15" origin="right"
      url="http://intranda.com/logo2.png"/>
   <text x="200" y="30" fontname="Arial" fonsize="16"
      fontcolor="555555" origin="center">Beispieltext</text>
</watermark>
```

Beispielkonfiguration eines Wasserzeichens

3.5 PDF-Titelseiten

Ein PDF-Dokument enthält eine oder mehrere Seiten eines oder mehrerer Dokumente. Ein PDF-Dokument kann somit aus Teilen mehrere Dokumente zusammengestellt werden. Sowohl zum Unterscheiden der einzelnen Teile als auch zum Identifizieren des gesamten PDF-Dokuments können optional PDF-Titelseiten eingefügt werden. Neben allgemeinen Informationen wie bspw. den Nutzungsbedingungen (Terms & Conditions) enthält die Titelseite auch Metadaten, die zur Identifizierung des Werks notwendig sind.

Die einzelnen Teile der Titelseite können dabei aus verschiedenen Quellen stammen. Allgemeine Informationen können in einer zentralen Konfigurationsdatei hinterlegt werden, während dokumentspezifische Informationen wie bspw. die Metadaten der METS-Datei entnommen werden können, welche zur Generierung der PDF-Datei genutzt wurde.

3.5.1 Komponenten einer PDF-Titlepage

Eine PDF-Titelseite besteht aus verschiedenen Komponenten. Wesentlicher Bestandteil sind die Metadaten. Diese werden in eigenen Textblöcken angezeigt, die über spezielle typographische Unterscheidungsmerkmale (Schriftart und Schriftgröße) verfügen. Theoretisch ist die Anzahl der Metadatentextblöcke unbegrenzt. Diese Daten werden oben

links auf die Seite geschrieben. Eine genaue Positionierung der Metadatentextblöcke ist nicht möglich.

Neben den Metadatentextblöcken können auch Nutzungsbedingungen definiert werden. Diese werden unterhalb der Metadatentextblöcke auf der Seite platziert. Ggf. können diese mit einer Überschrift versehen werden. Die Nutzungsbedingungen werden in einzelne Absätze unterteilt.

Für alle Textblöcke (Metadaten und Nutzungsbedingungen) gilt, dass diese automatisch umbrochen werden. Derzeit kann die Typografie sowohl der Nutzungsbedingungen als auch der Metadatentextblöcke nicht variiert werden. Sowohl Schriftart als auch Schriftgröße sind fest kodiert.

Images, die bspw. Logos beinhalten können, können frei auf der Seite platziert werden. Hierbei ist jedoch zu beachten, dass die übrigen Textblöcke, nicht um das Logo herum platziert werden.

3.5.2 Ursprung der Inhalte

Einzelne Bereiche der PDF-Titelseite können einen unterschiedlichen Ursprung haben. Während die Nutzungsbedingungen in einer zentralen Konfigurationsdatei festgelegt werden, kommen die Metadaten üblicherweise aus der METS Datei. Aus Gründen der Abwärtskompatibilität können die Metadaten auch in die zentrale Konfigurationsdatei für die Titelseite geschrieben werden. Dies kann vor allem dann sinnvoll sein, wenn die Konfigurationsdatei von externen Systemen dynamisch generiert wird. Dies ist möglich, da die Konfigurationsdatei sowohl aus einem Dateisystem als auch per http-Zugriff geladen werden kann. Der entsprechende Dateipfad bzw. die URL werden im Aufruf zur PDF-Generierung mit übergeben.

In aller Regel werden die Metadaten jedoch direkt der METS Datei entnommen. Eine entsprechende Klasse sorgt hierbei für die Extrahierung der Metadaten und konvertiert diese in Strings. Diese werden dann an die PDF-Konvertierung übergeben. Allerdings kennt die Schnittstelle lediglich 4 Metadatentextblöcke (line1 - line4). Ein Metadatentextblock kann dabei mehr als ein Metadatum enthalten. Vielfach sind die Metadaten auch abhängig von bestimmten Dokumenttypen, die in der METS Datei definiert sind. Eine Zeitschrift hat bspw. andere Metadaten als ein Kapitel einer Monographie. Wie diese Metadaten zusammengebaut werden, ist nicht konfigurierbar. Änderungen erfordern eine neue Implementierung der MetadataExtractor Klasse. Aufgrund der vielen möglichen Abhängigkeiten sind entsprechenden Konfigurationsmöglichkeiten zu aufwendig zu implementieren.

3.5.3 Konfiguration von PDF-Titelseiten

Die nachfolgend beschriebene Konfiguration bezieht also nur auf die allgemeine Konfiguration der PDF-Titelseite. Die Konfigurationsdatei sieht noch Felder für entsprechende Metadatentextblöcke vor. Diese Daten werden nur berücksichtigt, wenn keine Metadaten aus der METS Datei extrahiert werden können. Die MetadatenExtraktor Klasse muss in einem solchen Fall „null“ zurückliefern (siehe Abschnitt zur API Beschreibung).

Die allgemeine Konfiguration der PDF Titelseite wird in einer XML-Datei beschrieben. Als alles umfassendes Element (=root-Element) dient das Element <pdftitlepage>. Dieses Element kann folgende Unterelemente besitzen:

- *parenttype*: Dieses Element gibt den Dokumenttyp an (bspw. Zeitschrift etc.) und wird oberhalb der Metadaten geschrieben. Diese Information kann ebenfalls aus der METS Datei übernommen werden. Dieses Feld ist optional.
- *line1 – line4*: Diese 4 Elemente beinhalten die Metadatentextblöcke.
- *termsconditions*: beinhaltet verschiedene Absätze (<p> Elemente), welche den eigentlich Text der Nutzungsbedingungen enthalten. Nach jedem <p> Element wird eine neue Zeile begonnen. Die Überschrift der Nutzungsbedingungen ist im Attribut title enthalten.
- *image*: definiert ein Image mit entsprechender URL und Koordinaten. Die URL ist der Inhalt des Elements. Die Koordinaten sind im x und y Attribut gespeichert. Es handelt sich hierbei um die linke untere Ecke des Images. Die Koordinaten sind keine Pixelkoordinaten, sondern Punktkoordinaten (points), mit denen das PDF Format intern arbeitet. Optional kann noch ein Skalierungsfaktor für das Image angegeben werden, um den das Image verkleinert werden kann. Das scale Attribut enthält dazu einen Wert zwischen 0 und 1.

Aufbau der PDF-Titelseite

Das nachfolgende Beispiel zeigt eine Konfigurationsdatei für ein Wasserzeichen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pdftitlepage PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
   "http://dz-srv1.sub.uni-goettingen.de:8080/sub/pdftitlepage.dtd">
<pdftitlepage>
   <!-- usually line1 contains the title of the selected document -->
   <line1>Reformation in Hamburg</line1>
   <!-- line 2 contains the creator of the document -->
   <line2 type="parent">Leopold Kist</line2>
   <!-- line 3 contains the title of the parent,
        in which the selected document is contained -->
   <line3>in: Amerikanisches </line3>
   <!-- line4 contains the pages; from and until -->
   <line4>pp. 15 - 27</line4>
   <termsconditions title="Terms and Conditions">
      <p>The Goettingen State and University Library provides
         access to digitized documents ...
      </p>
      <p>For reproduction requests and permissions, please contact us.
         If citing materials, please give proper ...
      </p>
      <p>Contact:</p>
      <p>Niedersaechsische Staats- und Universitaetsbibliothek</p>
      <p>Digitalisierungszentrum</p>
      <p>37070 Goettingen</p>
      <p>Germany</p>
      <p>Email: gdz@www.sub.uni-goettingen.de</p>
   </termsconditions>
</pdftitlepage>
```

Beipielkonfiguration eine PDF-Titelseite

4. Nutzung der Servlets

Der ContentServer (CS) kann unter http://<Server Adresse>/cs/cs erreicht werden. Ein Aufruf könnte dann z.B. so aussehen:

```
http://intranda.com/cs/cs?action=echo
```

Contentserver Aufruf

Der Goobi ContentServer (GCS) kann unter http://<Server Adresse>/gcs/gcs erreicht werden. Hier sieht der besipielhafte Aufruf wie folgt aus:

```
http://intranda.com/gcs/gcs?action=echo
```

Goobi Contentserver Aufruf

Die Befehle werden über action=<Befehl> an den jeweiligen Server gerichtet. Über url-Parameter wird entsprechend der Befehl modifiziert.

Über action=help kann man sich jeweils die Hilfeseite anzeigen lassen, auf der sich unter anderem die möglichen Befehle und die nachfolgenden url-Parameter anzeigen lassen.

4.1 Erläuterung der URL-Parameter des ContentServers

Hier eine Auflistung der Beschreibungen aus der Hilfedatei für die unterschiedlichen Aufgaben des Content Servers sowie dessen mögliche url-Parameter:

4.1.1 action=image

By using the action "image" you can process some image manipulations like converting, scaling, rotating, etc. Simply call the URL: "http://intranda.com/cs/cs?action=image" and combine it with the necessary parameters. The following URL shows as example, how to convert a tif image into a jpeg image, scale it to 30% of its size, rotate it 90° and add two coloured highlighting areas:

```
http://intranda.com/cs/cs?action=image&sourcepath=00000001.tif&scale=30&rotate=0&format=jpeg&resolution=200&highlight=10,50,80,150
 60,80,160,200
```

Beispiel URL action=image

Parameter	Description
sourcepath	path and name of source image (repository path is configured for ContentServer)
rotate	value of the rotation angle as number between 0 and 360 sample: "rotate=90"
scale	value of zoom factor (for example: 50 represents 50% and 100 represents 100%) sample: "scale=50"

Parameter	Description
width	pixel for zoom to fixed width (for example: 800 represents 800px width) sample: "width=800"
height	pixel for zoom to fixed height (for example: 1000 represents 1000px height) sample: "height=1000"
format	format of target file, possible formats are: jpeg l png l jp2 l tiff
resolution	define resolution of target image in dpi (default is 600 dpi)
highlight	define coordinates for highlighting inside the image; define 4 coordinates for each highlighting area; for multiple highlighting areas separate the coordinates of each area using a pipe "l" sample for one area: "highlight=10,50,80,150" sample for two areas: "highlight=10,50,80,150l 60,80,160,200"
targetFileName	define the name of the target image file sample: "targetFileName=myGeneratedImage.jpg"
errorReport	format of the error report; possible values "image" and "jsp"; if parameter is not defined "jsp" is taken sample: "errorReport=image"

url-Parameter ContentServer

4.1.2 action=pdf

By calling the action "pdf" you can merge multiple image files together with some general metadata and bookmarks to one pdf file. Simply call the following URL: "http://intranda.com/cs/cs?action=pdf" and combine it with some parameters to generate your pdf file. The following URL is an example on how you can process eight images together with some author and title metadata and some bookmarks to generate one pdf file:

```
http://intranda.com/cs/cs?action=pdf&images=00000001.tif|00000002.tif|00000003.tif|00000004.tif|00000005.tif|00000006.tif|
00000007.tif|00000008.tif&imageNames=I|II|unnamed|1|2|3|4|5&bookmarks=1,0,0,Frontpage|2,0,1,First Act|3,2,1,Chapter
 One|4,2,2,Chapter Two|5,0,5,Second Act|6,5,6,Chapter Three|7,5,7,Chapter Four&metadataAuthor=Johann Wolfgang von
 Goethe&metadataTitle=Faust: The First Part of the Tragedy&metadataCreator=Mr. Goethes Hidden Creator&metadataSubject=classical
 literature&metadataKeyword=faust, first part&targetFileName=Goethe_Faust_FirstPart.pdf
```

Beispiel URL action=pdf

Parameter	Description
images	define the list of images to be transformed to one pdf file by using the path and name of each image file (repository path is configured for ContentServer) and separate all files by a pipe "l"

Parameter	Description
	sample for three images: "images=00000001.tifl 00000002.tifl00000003.tif"
imageNames	define the list of names for all images to be set for the pdf file by using the name for each image file of the defined images and separate all names by a pipe "l" sample for three image names: "imageNames=Ilal 3"
bookmarks	define the bookmarks as list using pipes "l" as separator. Each bookmark consists of four properties which are separated by a comma: "id, parentId, index of image in image list starting by 0, title of bookmark". By choosing 0 as parentId, the bookmark is set to the top level. Each ID has to be unique. All referenced images index numbers have to exist in the list of images. sample for four bookmarks: "bookmarks=1,0,0,Frontpagel2,0,1,"First Actl 3,1,1,Chapter Onel4,1,14,Chapter Two"
metadataAuthor	define the metadata for the author sample: "metadataAuthor=Johann Wolfgang von Goethe"
metadataCreator	define the metadata for the creator sample: "metadataCreator=Mr. Goethes Hidden Creator"
metadataTitle	define the metadata for the title sample: "metadataTitle=Faust: The First Part of the Tragedy"
metadataSubject	define the metadata for the subject sample: "metadataSubject=classical literature"
metadataKeyword	define the metadata for the keyword sample: "metadataKeyword=faust, first part"
targetFileName	define the name of the target pdf file sample: "targetFileName=Goethe_Faust_FirstPart.pdf"
alwaysUseRenderedImage	convert all images always to rendered image before sending it to iText; if parameter is not defined the value ist taken from config file sample: "alwaysUseRenderedImage:false"
alwaysCompressToJPEG	compress all images always to jpeg image before sending it to iText; if parameter is not defined the value ist taken from config file sample: "alwaysCompressToJPEG:false"

Parameter	Description
errorReport	format of the error report; possible values "image" and "jsp"; if parameter is not defined "jsp" is taken sample: "errorReport:image"

Neue Tabelle

4.2 Erläuterung der URL-Parameter des Goobi Content Servers

Eine Auflistung aller nutzbaren Parameter für die Verwendung des Goobi Content Servers findet hier:

4.2.1 action=pdf

By calling the action "pdf" you receive one pdf file on basis of a given mets file, including page names, bookmarks, watermarks and title pages. Simply call the following URL: "http://intranda.com/gcs/gcs?action=pdf" and combine it with some parameters to generate your pdf file. The following URL is an example on how you can process one mets file and generate one pdf file from it:

```
http://intranda.com/gcs/gcs?action=pdf&metsFile=FaustTragedy.xml&targetFileName=Goethe_Faust_FirstPart.pdf
```

Beispiel URL action=pdf

Parameter	Description
metsFile	define the name of the METS file. Use here only the relative path coming from the defined repository. sample: "metsFile=FaustTragedy.xml"
targetFileName	define the name of the target pdf file sample: "targetFileName=Goethe_Faust_FirstPart.pdf"
divID	define the mets div by id which should be generated as pdf file sample: "divID=log9"
writeAsPdfA	define if the pdf should be written as Pdf/A file sample: "writeAsPdfA=false"
metsFileGroup	define which metsFileGroup should be used sample: "metsFileGroup=PRESENTATION"
pagesize	define which pagesize should be used ('A4', 'original', 'A4Box sample: "pagesize=A4"
pdftitlepage	if the content of the metsfile should not be parsed automatically you can define here a title page for the PDF-file. Its content will be used directly

Parameter	Description
	without any change. Be sure to use an url as parameter here. sample: "pdftitlepage=http://intranda.com/GDZ/samplepdftitlepage.xml"
ignoreCache	define here if a pdf file from cache should be ignored sample: "ignoreCache=true"

url-Parameter action=pdf

4.2.2 action=metsImage

By calling the action "metsImage" you receive one image file from a given mets file with the given divID which can be processed by the embedded contentSever image library. Simply call the following URL: "http://intranda.com/gcs/gcs?action=metsImage"" and combine it with some parameters to process the requested image file. The following URL is an example on how you can process one image from a mets file:

```
http://intranda.com/gcs/gcs?action=metsImage&metsFile=FaustTragedy.xml&divID=phys2&scale=30&rotate=90&format=jpeg&resolution=200
```

Beispiel action=metsImage

Parameter	Description
metsFile	Define the name of the METS file. Use here only the relative path coming from the defined repository. sample: "metsFile=FaustTragedy.xml"
targetFileName	define the mets div id of the requested image which should be processed, the id have to be a physical one sample: "divID=phys2"
divID	define which metsFileGroup should be used; the image is taken from this file group; if not defined, the value from configuration file will be used sample: "metsFileGroup=PRESENTATION"
sourcepath	path and name of source image (repository path is configured for ContentServer)
rotate	value of the rotation angle as number between 0 and 360 sample: "rotate=90"
scale	value of zoom factor (for example: 50 represents 50% and 100 represents 100%)

Parameter	Description
	sample: "scale=50"
width	pixel for zoom to fixed width (for example: 800 represents 800px width) sample: "width=800"
height	pixel for zoom to fixed height (for example: 1000 represents 1000px height) sample: "height=1000"
format	format of target file, possible formats are: jpeg l png l jp2 l tiff
resolution	define resolution of target image in dpi (default is 600 dpi)
highlight	define coordinates for highlighting inside the image; define 4 coordinates for each highlighting area; for multiple highlighting areas separate the coordinates of each area using a pipe "l" sample for one area: "highlight=10,50,80,150" sample for two areas: "highlight=10,50,80,150l60,80,160,200"
targetFileName	define the name of the target image file sample: "targetFileName=myGeneratedImage.jpg"
errorReport	format of the error report; possible values "image" and "jsp"; if parameter is not defined "jsp" is taken sample: "errorReport=image"

url-Parameter action=metsImage

4.2.3 action=multiPdf

By calling the action "multiPdf" you receive one pdf file from a given list of mets and pdf files. Simply call the following URL: "http://intranda.com/gcs/gcs?action=multiPdf" and combine it with some parameters to process the given files. The following URL is an example on how you can process one mets file and one pdf file into one merged pdf file:

```
http://intranda.com/gcs/gcs?action=multiPdf&files=mets|someMetsFile.xml||pdf|somePdfFile.pdf&metsFileGroup=DEFAULT
```

Beispiel URL action=multiPdf

Parameter	Description
files	As filetype you can use 'pdf' and 'mets'.

Parameter	Description
	The url parameter defines the name of the PDF or METS file. Use here only the relative path coming from the defined repository. As divid you can insert a physical mets div id. sample: "files=mets|someMetsFile.xml||pdf|somePdfFile.pdf||mets|someMetsFileWithDivId.xml|phys2"
targetFileName	define the name of the target image file sample: "targetFileName=merged.pdf"
divID	define the mets div by id which should be generated as pdf file sample: "divID=log9"
writeAsPdfA	define if the pdf should be written as Pdf/A file sample: "writeAsPdfA=false"
metsFileGroup	define which metsFileGroup should be used; the image is taken from this file group; if not defined, the value from configuration file will be used sample: "metsFileGroup=PRESENTATION"
pagesize	define which pagesize should be used ('A4', 'original', 'A4Box sample: "pagesize=A4"

url-Parameter action=pdf

5. Nutzung der API

Der ContentServer basiert auf Klassen, die bestimmte Funktionalitäten des ContentServer kapseln. Diese Klassen lassen sich auch außerhalb des Servletkontext bspw. für eigene Projekte nutzen. Daher werden die Klassen auch als API bezeichnet, auch wenn sie keine Programmierschnittstelle für den ContentServer sind. Vielmehr nutzen sie Klassen, auf denen auch der ContentServer basiert. Dies hat zur Folge, dass die Funktionalität des ContentServers auf eine Vielzahl von Klassen verteilt ist. Einen genauen Überblick über die einzelnen Klassen und Methoden gibt die javadoc-Dokumentation. Dieser Abschnitt soll nur einen kleinen Überblick über die Klassen geben.

Es empfiehlt sich ohnehin mit einigen der Klassen vertraut zu machen, Der ContentServer wurde auf Basis einer Architektur entwickelt, die auf eine Nachnutzbarkeit und Erweiterbarkeit ausgelegt ist. So lässt sich der ContentServer durch das Ersetzen bestimmter Klassen und durch die Implementierung standardisierter Java-Interfaces um wesentliche Funktionen gerade hinsichtlich der Metadaten erweitern.

Ferner vermittelt dieser Abschnitt ein grundlegendes Verständnis, wie der ContentServer intern arbeitet und welche Änderungen notwendig wären, um bspw. zusätzliche Imageformate zu unterstützen.

5.1 Aufbaus der Api

Der ContentServer und damit die API bestehen aus drei grundlegenden Teilen, deren Interfaces aufeinander abgestimmt sind. Instantiierte Objekte lassen sich zwischen den einzelnen Teilen ohne großen Aufwand austauschen. Der Servicelayer nutzt diese Teile ebenfalls. Es ist daher möglich, dass sich die API aufgrund von geänderten Anforderungen und Erweiterungen am ContentServer ändert.

5.1.1 ImageLib

Die ImageLib enthält alle Klasse zur Imagebearbeitung. Dazu zählt die Imagebearbeitung im engeren Sinn als auch die Klassen zum Einladen und Speichern der Imageformate. Die Imagebearbeitung selbst konzentriert sich derzeit auf das Skalieren von Images. Intern arbeiten alle Klasse der ImageLib mit Objekten der RenderedImage Klasse. Dieses dient als Übergabeparametern zwischen den Klassen. Ein RenderedImage enthält lediglich Pixeldaten sowie die wesentlichsten Metadaten (Höhe/Breite) eines Images. Weitere Metadaten können nur beim Lesen (und eingeschränkt auch beim Schreiben) berücksichtigt werden.

5.1.2 ImageInterpreter

Die ImageLib verfügt über verschiedene Image-Interpreter. Für jedes Dateiformat muss eine Klasse vorhanden sein, die das ImageInterpreter Interface implementiert. Derzeit kennt der ContentServer folgende Image Interpreter:

- *PNG*
- *TIFF*
- *JPG*
- *JPEG2000*

Image Interpreter

Instantiiert wird eine entsprechende ImageInterpreter Klasse durch den ImageManager. Dieser sorgt dafür, dass ein InputStream und ein OutputStream bereitstehen. Jeder ImageInterpreter selbst ist unabhängig von Eingabe und Ausgabedateien implementiert. Ihm müssen lediglich InputStream und OutputStream übergeben werden. Der ImageManager ist ebenfalls dafür verantwortlich, den entsprechenden Interpreter zu bestimmen. Die entsprechenden Methoden stellt die ImageFileFormat Klasse bereit, um das Imageformat anhand des MIME Typs (für http-URLs) oder der Dateiendung zu bestimmen. Derzeit ist keine Möglichkeit vorgesehen, die Formatunterstützung extern zu verwalten. Entsprechende Klassen, die das ImageInterpreter Interface implementieren können zwar erstellt werden. Jedoch ist für die Einbindung der Klassen eine Ergänzung der Klasse ImageFileFormat notwendig.

Jede Klasse, die das ImageInterpreter Interface implementiert muss Methoden zum Lesen und Schreiben des Images aus/in Stream bereitstellen. Welche zusätzliche Metadaten außer den Dimensionen eines Images dazu unterstützt werden müssen sind nicht vorgegeben. Die Klasse AbstractImageInterpreter von dem alle anderen ImageInterpreter Klassen abgeleitet sein sollten, implementiert einige Dummy-Methoden, die in den jeweiligen ImageInterpreter Klassen überladen werden können. Eine weitere wesentliche Methode jeder ImageInterpreter Klasse ist das rendern des jeweiligen Imageformats in ein RenderedImage. Diese dient zum Auslesen der Pixeldaten und wird von anderen Klassen wie dem ImageManager oder dem ImageManipulator genutzt.

5.1.3 ImageManager

Der ImageManager stellt alle Methoden zum Skalieren von Images zur Verfügung. Als Schnittstelle zu anderen Klassen dient dabei eine RenderedImage-Instanz.

Intern bedient sich der ImageManager der ImageManipulator Klasse. Diese ist jedoch nicht für die Verwendung außerhalb des ContentServers freigegeben. Sämtliche Image Manipulationen sollten immer über den ImageManager geschehen. Die ImageManager Klasse kapselt die autonomen Aufrufe aus der ImageManipulator Klasse. Eine einfache

Skalierung kann u.U. nicht nur die Skalierung des Images bedeuten, sondern muss auch Koordinaten der Wörter, die farbig hinterlegt werden sollen (highlighting) entsprechend skalieren. Die ImageManipulator Klasse stellt entsprechende Methoden hierzu zur Verfügung, die von der ImageManager Klasse genutzt werden.

Die ImageManager Klasse wird immer unter Angabe einer URL instantiiert. Die URL gibt dabei die Position des Images an. Der Kontext einer ImageManager Instanz ist immer ein Image. Soll ein weiteres Image skaliert werden, muss dazu eine weitere ImageManager Instanz initiiert werden. Derzeit werden http- und file-URLs unterstützt.

Der ImageManager lädt das entsprechende Image sofort nach Instanziierung und stellt ein entsprechendes ImageInterpreter Objekt bereit.

```
URL tifurl = new URL(basefolder.toString() + "testdata/gauswerk_PPN236018647_0010_01_tif/00000001.TIF");
ImageManager sourcemanager = new ImageManager(tifurl);
ImageInterpreter si = sourcemanager.getMyInterpreter;
```

ImageInterpreter

Über das ImageInterpreter Objekt kann dann auch auf das RenderedImage sowie auf die Metadaten des Images zugegriffen werden.

Die zentrale Funktion der ImageManager Klasse ist die Methode *scaleImageByPixel*. Diese Methode existiert in verschiedenen Ausführungen, die allerdings lediglich verschiedene Komplexitäten der Skalierung erlauben. Sie unterscheiden sich daher nur in der Anzahl der Parameter. Intern finden immer die gleichen Skalierungsmethoden aus der ImageManipulator Klasse Anwendung.

```
public RenderedImage scaleImageByPixel(int pixelx,int pixely, int scaleby, int angle,
                                       LinkedList<String> coordinates, Color inColor)
```

Methode scaleImageByPixel

Die hier beschriebene Methode stellt die maximale Funktionalität bereit. Im Folgenden sind ihre Parameter beschrieben:

- *pixelx* Skalierungsfaktor in X-Richtung. Entweder die Breite des Zielimages oder Faktor in Prozent.
- *pixely* Skalierungsfaktor in Y-Richtung. Entweder die Höhe des Zielimages oder Faktor in Prozent.
- *scaleby* bestimmt, ob auf eine feste Breite skaliert wird oder die Angaben als Prozentangaben zu verstehen sind. Enthält entweder den Wert ImageManager.SCALE_BY_PERCENT oder ImageManager.SCALE_BY_WIDTH.
- *angle* Grad für Rotation des Images; Es werden nur ganzzahlige Angaben zwischen 0 und 360 Grad interpretiert. Wird das Image rotiert (d.h. weicht die Angabe von 0 ab), werden derzeit keinerlei Koordinaten gerendert (d.h. highlighted).
- *coordinates* Enthält eine Liste von Strings, welche Koordinaten eines Rechtecks im Format „x-oben-links, y-oben-links, x-unten-rechts, y-unten-rechts“ enthält. Diese Box

wird entsprechend highlighted. Das Format des Strings wird zukünftig für beliebige Polygone ausgebaut werden.
inColor ein Java Color-Objekt, welches die Farbe für das Highlighting bestimmt.

Parameter scaleImageByPixel

Die Methode *scaleImageByPixel* liefert immer ein RenderedImage zurück oder liefert eine *ImageManipulatorException*. Das RenderedImage wird üblicherweise einer neuen ImageInterpreter Instanz übergeben. Diese Instanz wird dann dazu genutzt, das Image in einen OutputStream zu schreiben. Hierbei ist der Aufrufer für die gewünschte Übertragung von Imagemetadaten selbst verantwortlich.

Die Methode kann um Watermarks ergänzt werden, die nach der Skalierung dem RenderedImage hinzugefügt werden. Folgende zusätzliche Parameter müssen dazu in der folgenden Reihenfolge angeben werden:

```
public RenderedImage scaleImageByPixel(int pixelx,int pixely, int externalscalemethod, int angle, LinkedList<String> coordinates,
 Color inColor, Watermark inWatermark, boolean watermarkscale,int watermarkposition) throws ImageManipulatorException
```

Methode scaleImageByPixel (mit Watermark)

- *inWatermark* ein Wasserzeichenobjekt, welches dem skalierten Image angefügt wird. Nachfolgende Parameter bestimmen die Art und Weise wie dies geschieht.
- *watermarkscale* bestimmt wie das Wasserzeichen skaliert wird. Wird als Parameter true übergeben wird das Wasserzeichen nach dem rendern proportional skaliert. Andernfalls wird es nur entsprechend der Breite oder der Höhe angepasst wobei die Ausrichtung der internen Komponenten berücksichtigt wird.
- *watermarkposition* legt die Position des Wasserzeichens im Verhältnis zum Image fest. Mögliche Werte sind: *ImageManager.BOTTOM*, *ImageManager.TOP*, *ImageManager.LEFT*, *ImageManager.RIGHT*.

Parameter scaleImageByPixel (mit Watermark)

Die interne javaDoc Dokumentation sowie die Beispiele in den Testklassen geben weitere Auskünfte über Wasserzeichen und wie diese instantiiert werden.

5.1.4 PDFManager

Die PDFManager erlaubt die Erstellung von PDF-Dokumenten auf Basis von Images. Sie nutzt dazu auch Klassen aus der ImageLib um bspw. entsprechende Images einzuladen. Intern dient hierbei die ImageInterpreter Klasse als Parameter zwischen dem PDFManager und der ImageLib, da sich so die für die PDF Erstellung wichtigen technischen Metadaten (Auflösungsinformation) eines Images übergeben lassen.

5.1.5 PDF-Generierung

Der PDFManager wird pro PDF Dokument instantiiert. Der einzige verpflichtende Parameter für die Generierung eines PDFs ist eine HashMap mit URLs. Jede Seite

bekommt eine interne Seitennummer (integer Wert), der auch zur Bestimmung der Reihenfolge der Seiten dient. Diese Seitennummer gleichzeitig der Schlüssel in der HashMap in denen die URLs gespeichert werden. Eine entsprechende HashMap kann bereits im Konstruktor der PDFManager Klasse übergeben werden.

Das eigentliche PDF wird durch die Methode createPDF erzeugt. Neben Parameter zur Bestimmung der Seitengröße muss als weiterer Parameter ein OutputStream übergeben werden. In diesen OutputStream wird der Bytestream des PDFs während der Generierung hineingeschrieben. Das PDF wird intern also nicht erstmal komplett generiert und dann an den OutputStream ausgeliefert. Kommt es daher während der Verarbeitung intern zu Fehlern, resultiert dies in einem unvollständigen OutputStream. Zusätzlich wird eine Exception geworfen. Alle Einstellung der PDF Generierung betreffend müssen mittels entsprechender Setter vor dem Aufruf der createPDF Methode gemacht werden. Sobald die Methode läuft, können keine weiteren Einstellungen vorgenommen werden.

Als Seitengröße kennt die PDFManager Klasse folgende Möglichkeiten:

- *PDF_ORIGPAGESIZE* Die Seitengröße ist in Originalgröße der Vorlage
- *PDF_A4PAGESIZE* Die Seite ist DIN A4 groß. Die Vorlage wird horizontal zentriert auf der Seite platziert. Zu große Vorlagen werden
  auf DIN A4 verkleinert. Vorlagen im Querformat werden automatisch um 90 Grad rotiert, um die DIN A4 Seitengröße besser auszurichten.
- *PDF_A4PAGESIZE_BOX* Wie oben, jedoch wird um die Vorlage ein kleiner schwarzer Rahmen gezogen.

PDF-Einstellungen

5.1.6 Bookmarks

Der PDFManager kann optional Bookmark-Objekte dazu nutzen, um eine hierarchische Navigation im Dokument zu ermöglichen. Dazu können Bookmark Objekte weitere Bookmark beinhalten. Jedes Bookmark Objekt hat neben eines Textes (enthält üblicherweise die Überschrift eines Kapitels, Artikels etc.) auch eine Seitennummer. Diese Seitennummer muss mit der internen Seitennummer in der HashMap in der auch die Image-URLs abgelegt werden übereinstimmen, damit für die entsprechende Seite ein Bookmark angelegt wird.

Prinzipiell ist es möglich, dass mehrere Root-Bookmark Elemente existieren. Alle Wurzelelemente müssen in einer Liste enthalten und dem PDFManager mittels der Methode *setRootBookmarkList* übergeben werden. Beim Aufruf der *createPDF* Methode werden diese dann automatisch berücksichtigt.

5.1.7 Seiten-Labels

Seiten haben in aller Regel so genannte Labels. Diese Labels können bspw. Seitennummer oder einen kurzen beschreibenden Text beinhalten. Im Gegensatz zu Bookmarks sind Seiten-Labels nicht hierarchisch, sondern in einer flachen HashMap organisiert. Der Schlüssel in der HashMap ist wieder die interne Seitennummer (Integer Wert). Dieser ordnet den Text der entsprechenden Seite bzw. dem entsprechenden Seitenimage zu.

Seitenlabels können optional angelegt werden. Sie werden dem PDFManager als HashMap mittels der Methode setImageNames übergeben.

5.1.8 METS Parser

Grundlage für die PDF Generierung im ContentServer ist eine METS Datei. Eine solche METS Datei kann entweder aus einem Dateisystem oder mittels einer http-URL geladen werden. Die METS Datei muss dem DFG-Viewer Profil entsprechen. Somit enthält die METS Datei neben deskriptiven Metadaten auch eine Liste von Imagedateien sowie entsprechende Strukturdaten.

Aus diesen Daten werden alle Daten extrahiert, die der PDF-Manager benötigt, um eine PDF-Datei zu generieren. Optional können auch die Metadata extrahiert werden, um eine PDF-Titelseite zu erzeugen.

Die Klasse METSParser benötigt als Parameter im Konstruktor die URL zu der METS-Datei. Diese wird beim instantiieren eines METSParser Objektes angelegt. Die METSParser Klasse stellt anschließend Methoden zur Verfügung, um auf die Inhalte der METS Datei zuzugreifen. Einige Methoden erlauben den Zugriff auf einzelne Elemente innerhalb der Datei. Da der METS-Parser auf die xmlbeans aufsetzen, können sich Methoden, die Objekte aus den xmlbeans-Klassen liefern oder übernehmen mit der Zeit ändern. Die derzeit einzig für die Nutzung außerhalb der API freigegebenen Methoden sind getRootbookmarklist, getPageUrls und getPageNames. Diese Methoden liefern die für den PDF-Manager wichtigen Datenstrukturen (HashMap und Bookmark-Liste) zurück, so dass diese direkt an eine PDFManager Instanz übergeben werden können.

5.2 Aufbau der API - Anpassungen durch Implementierung von Interfaces

Der ContentServer wurde in weiten Teilen konfigurierbar gestaltet. Die Konfiguration erfolgt dabei entweder über Parameter bei seinem Aufruf oder durch XML basierte Konfigurationsdateien. Sie definieren mögliche Verhaltensweisen des ContentServers. Da sich die zukünftige Nutzung sowie Datenformate stark von den heutigen unterscheiden können, wurden sowohl die ImageLib als auch der PDF-Manager wurden in eine

nachhaltige Architektur eingebunden. Diese erlaubt eine weitergehend Anpassung durch die Implementierung neuer Klassen. Einzige Voraussetzung an diese neuen Klassen ist, dass sie ein spezielles Interface implementieren müssen.

5.2.1 ImageInterpreter

Mit der Zeit werden neue Dateiformate für Images unterstützt werden müssen. Dazu bedarf es einer entsprechenden Klasse, die das ImageInterpreter Interface implementiert. Bei der derzeitigen Struktur sind ferner Erweiterungen in der Klasse ImageFileFormat der ImageLib notwendig. Die entsprechenden neuen Dateiformate müssen der Methode getInterpreter bekannt gemacht werden, die für die Instanziierung entsprechender ImageInterpreter Klassen zuständig ist.

Der Aufwand für diese Änderungen ist vertretbar, vor allem vor dem Hintergrund, dass zusätzliche Dateiformate nur sehr selten hinzugefügt werden müssen.

5.2.2 MetadataExtractor

Eine Klasse, die das MetadataExtractor Interface implementiert ist dafür verantwortlich, deskriptive Metadata aus einer METS Datei zu extrahieren, um sie dem PDF-Manager bereitzustellen. Dazu kann eine solche Klasse auf den METS Parser selbst zugreifen, um von dort Metadaten aus mehreren Metadatensätzen zusammenzufügen.

Die Ergebnisse stellt die Klasse der PDFManager Klasse über folgende Methoden zur verfügen:

getPdf_titlepage_line1 - *getPdf_titlepage_line4*:

Diese Methoden liefern die Zeilen 1 – 4 für die PDF Titelseite als String. Der Inhalt des Strings ist nicht vorgeschrieben und kann entsprechend des Dokumenttyps variieren. Wird anstatt eines Strings „null“ zurückgeliefert, übernimmt die PDFManager Klasse die Daten aus der Konfigurationsdatei für die PDF-Titelseite.

getPdfcreator

Die Methode liefert einen String zurück, der Namen aller Personen enthält, die das Dokument erstellt haben. Der Inhalt des Strings wird durch die PDFManager Klasse in die PDF Metadatenfelder geschrieben.

getPdfkeywords

Die Methode gibt einen String zurück, der alle Keywords eines Dokuments enthält. Diese schreibt die PDFManager Klasse in die PDF-Metadatenfelder.

getPdftitle

Die Methode liefert den Titel des Dokuments als String zurück. Dieser kann neben dem Haupttitel auch ggf. zusätzliche Untertitel enthalten. Der Inhalt des Strings wird durch die PDFManager Klasse in die PDF Metadatenfelder geschrieben.

getStructType

Die Methode liefert den Typ des Dokuments als String. Er wird durch die PDFManager Klasse auf der PDF-Titelseite genutzt wird und kann alternativ auch in der Konfigurationsdatei zur PDF-Titelseite im Element <parenttype> angegeben werden.

Eine entsprechende Klasse, die das MetadataExtractor Interface implementiert muss dem METSParser Objekt mittels der Methode *setMetadataextractor* übergeben werden. Ihre calculatePDFMetadata Methoden wird anschließend während des Parsens der METS Datei aufgerufen.

5.2.3 BookmarkMetadataExtractor

Eine Klasse, die das BookmarkMetadataExtractor Interface implementiert ist verantwortlich für das Erstellen des Titels eines Bookmarks. Dies ist üblicherweise der Titel oder der Typ einer Struktureinheit (eines Elements) in der METS Datei. Das eigentliche Bookmark-Objekt wird nicht durch das Interface definiert.

Die *getBookmarkMetadata* Methode ist als einzige Methode in dem Interface definiert.

Eine entsprechende Klasse, die das BookmarkMetadataExtractor

Interface implementiert muss dem METSParser Objekt mittels der Methode *setBookmarkmetadataextractor* übergeben werden. Ihre *getBookmarkMetadata* Methode wird anschließend während des Parsens der METS Datei aufgerufen.