jadice 4.2.

jadice document platform

Michael Grossmann
Carolin Köhler

Februar 2009

# Annotationen

## Laden - Speichern - Bearbeiten

## Version 4.2.x

Dokumentation
für Entwickler

levigo solutions gmbh
Max-Eyth-Straße 30
D-71088 Holzgerlingen
Telefon: 07031 / 4161-0
Telefax: 07031 / 4161-50
eMail: info@levigo.de

# Inhaltsverzeichnis

# 1. Allgemeines

Der hier vorliegende Leitfaden führt in die technischen Zusammenhänge zwischen der jadice® Java Imaging Viewer Technology und Annotationen (zusätzlichen Dokument-Informationen) ein.

Die Dokumentation beschränkt sich im Wesentlichen auf die Bereiche, die für Entwickler interessant sind, um Annotationen mit jadice® programmtechnisch bearbeiten zu können und ist als Erweiterung zu der jadice viewer Integrations-Dokumentation zu verstehen.

Zur besseren Lesbarkeit werden Paketnamen nur in Fußnoten voll qualifiziert dargestellt.

Eine API-Referenz und eine Integrations-Dokumentation des jadice viewer werden jeweils als separate Dokumente zur Verfügung gestellt.

# 2. Einleitung

Unter Annotationen werden im jadice viewer

- Kommentare
- Vermerke
- Anmerkungen
- Erläuterungen
- Notizen oder
- Hinweise durch Pfeile oder farblich hinterlegte Bereiche

verstanden, die der Benutzer in einem Dokument auf einer bestimmten Seite anbringen kann. Annotationen sind zusätzliche Informationen zu einem Dokument und verändern das eigentliche Dokument nicht.

Diese Annotationen können Informationen in Form von

- Text oder
- grafischen Objekten zur
  - Verdeutlichung
  - Hervorhebung oder gar zur
  - Ausblendung

beinhalten und werden in einer eigenen Schicht „über" dem Dokument dargestellt.

Zur Zeit unterstützt der jadice viewer:

- IBM ContentManager 7.x und 8.x kompatible Annotationen als MO:DCA Strukturen.
- FileNet / FileNet P8 Annotationen als XML-Strukturen.
- zur reinen Anzeige Wang-Annotationen.

Ein gemischter Einsatz von Annotationsstrukturen ist nur beschränkt möglich.

# 3. Annotationstypen

Annotationen sind Dokument- bzw. Seiteninformationen, die in verschiedenen Ausprägungen im jadice viewer unterstützt werden.

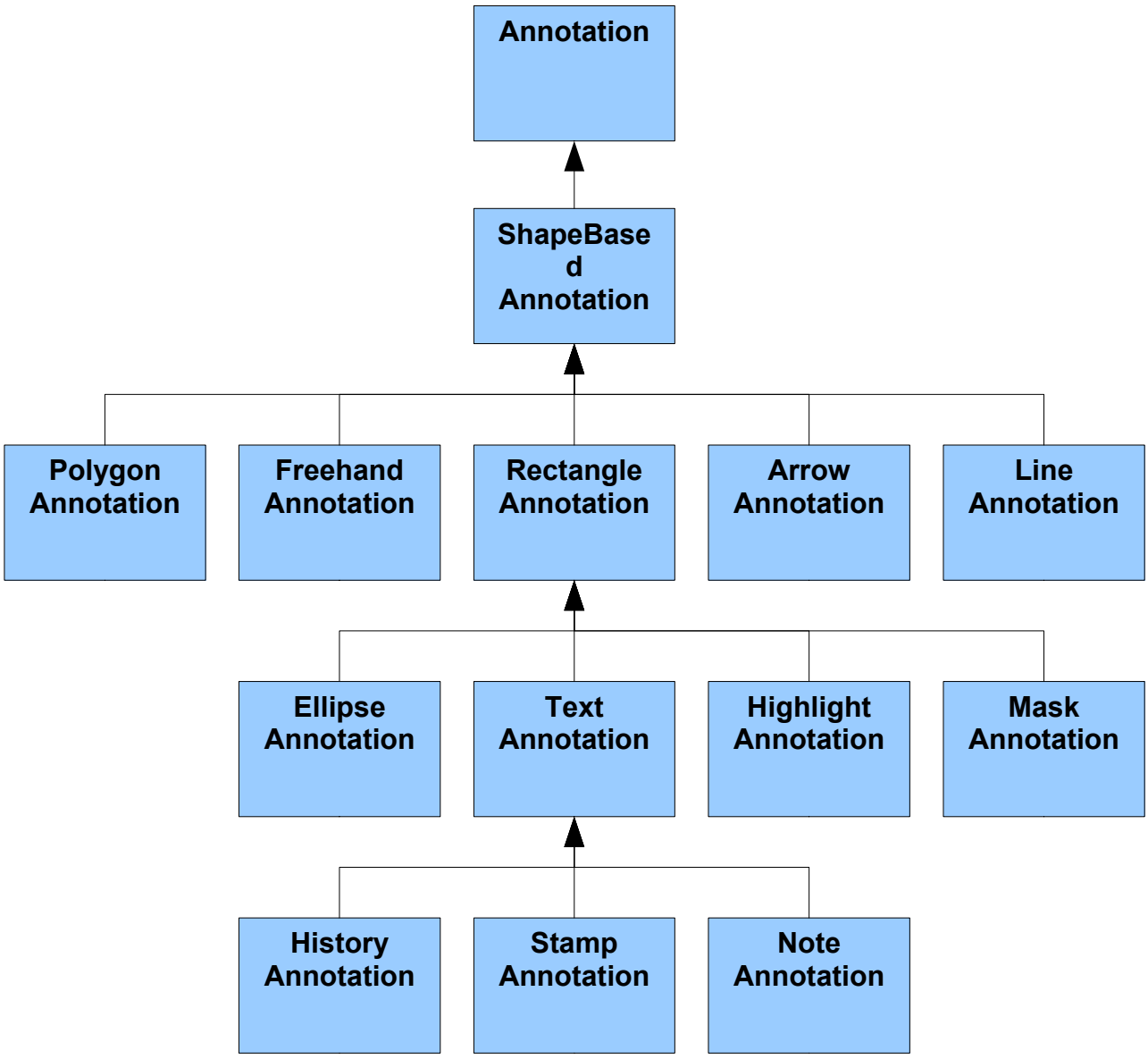


*Abbildung 1 - Annotation Architektur - Grobe Übersicht*

Zur Verfügung gestellte Typen sind:

- NOTE: eine Haftnotiz / Sticky
- HIGHLIGHT: eine Hervorhebung durch ein gefülltes und transparentes Rechteck
- MASK: eine Maskierung durch ein gefülltes, nicht transparentes Rechteck

- ARROW: ein Pfeil
- ELLIPSE: eine nicht gefüllte Ellipse
- RECTANGLE: ein nicht gefülltes Rechteck
- LINE: eine Linie
- TEXT: eine mit transparentem Hintergrund direkt auf der Seite aufgebrachte Textbemerkung
- STAMP: ein „Stempel" mit transparentem Hintergrund, Rahmen, rotierbar mit Textinhalt
- FREEHAND: eine Freihandzeichnung
- HISTORY: ähnlich Note, revisionssicher, gespeicherte Daten können nicht mehr gelöscht werden.
- POLYGON: Punkte, die mit Linien verbunden sind.

In Abbildung 1 sehen Sie einen groben Überblick der Klassen-Architektur von Annotationen. ShapeBasedAnnotation[1] ist eine abstrakte Basisklasse, die allen Annotationen die Eigenschaft gibt, eine geometrische Form (Shape[2]) zu haben. Als direkte Annotation ist diese Klasse nicht von Relevanz, für Integratoren jedoch, die eigene Annotationen definieren möchten, stellt sie eine nützliche Basis dar.

Im Gegensatz zur Viewer Generation 2.x stand eine logische Vererbung, aufgrund der Darstellungs- und Verarbeitungs-Eigenschaften einzelner Annotationstypen, im Vordergrund. Damit konnten zum einen Konsistenz und Vermeidung von Redundanzen gewährleistet werden, zum anderen ist ein flexibler Austausch des Annotations-Support für verschiedene Archivsysteme oder komplett selbst definierte Annotationen möglich geworden.

Für Entwickler, die die Annotations-Verwaltung nicht dem Viewer überlassen, sondern programmtechnisch Annotationen erstellen oder verändern wollen, bedingt diese Freiheit jedoch ein gewisses Maß an Vorsicht, um eine Kompabilität z.B. zu IBM ContentManager Annotationen zu gewährleisten.

## 3.1. VisualInfo / ImagePlus Annotationen

Annotationstypendefinition (jadice.viewer.annotation.type) in der Jadice.properties Datei muss auf **vi** gesetzt werden !

ImagePlus ist mit folgenden Annotationen kompatibel:

- Highlight
- Mask
- Note

VisualInfo ist mit folgenden Annotationen kompatibel:

- Arrow
- Ellipse

1 com.levigo.jadice.annotation.ShapeBasedAnnotation
2 java.awt.Shape

- Freehand
- Highlight
- Line
- Note
- Rectangle
- Stamp
- Text

Zusätzliche Annotationstypen:

- History

  Hierbei handelt es sich um eine revisionssichere Annotation. Nach dem Speichern und erneutem Laden der Annotation kann der Text nicht mehr geändert/gelöscht werden. Diese Annotation kann in einem VisualInfo Client angezeigt werden, hier kann jedoch der Text geändert / gelöscht werden. Die Revisionssicherheit wird nur im jadice viewer unterstützt.

## 3.2. FileNet Annotationen

Annotationstypendefinition (jadice.viewer.annotation.type) in der Jadice.properties Datei muss auf **fn** gesetzt werden !

FileNet ist mit folgenden Annotationen kompatibel:

- Arrow
- Freehand
- Highlight
- Note
- Stamp
- Text

Nicht kompatible Annotationen werden nicht gespeichert. Es wird ein Warning in der Log-Ausgabe angezeigt.

## 3.3. FileNet P8 Annotationen

Annotationstypendefinition (jadice.viewer.annotation.type) in der Jadice.properties Datei muss auf **fnp8** gesetzt werden !

Zusätzlich sollte noch der Parameter `jadice.viewer.default-page-resolution=100` gesetzt werden.

Der Parameter definiert die Auflösung für Dokumente ohne eigene Auflösung. Dies ist nötig, um die Kompatibilität zum FileNet Viewer zu gewährleisten. Da die Annotationsgrössen anhand der Dokumentauflösung berechnet werden, kann es sonst zu einer fehlerhaften Darstellung kommen.

FileNet P8 ist mit folgenden Annotationen kompatibel:

- Highlight

- Note
- Arrow
- Ellipse
- Freehand
- Line
- Rectangle
- Stamp
- Text
- Polygon

Nicht kompatible Annotationen werden nicht gespeichert. Es wird ein Warning in der Log-Ausgabe angezeigt.

# 4. Berechtigung

Annotationen können definierte Zugriffsberechtigungen zugewiesen werden, die das Verhalten einer Annotation beim Anzeigen im Viewer und beim Speichern/Schreiben beeinflussen können.

Um Berechtigungen zu setzten / überprüfen, wird die AnnotationPermission[3] Klasse verwendet. Hier sind alle Berechtigungen und Methoden zum Ermitteln der Annotationsberechtigung definiert.

Folgende Berechtigungen und Kombinationen aus diesen sind möglich:

- PERMISSION_NONE

  keine Berechtigung, Annotation wird nicht angezeigt und kann nicht geändert, gelöscht oder gespeichert werden.

- PERMISSION_READ

  Annotation wird im Viewer angezeigt.

- PERMISSION_WRITE

  Annotation kann gespeichert werden.

- PERMISSION_CHANGE

  Annotation kann verändert werden (nicht gelöscht !!!),

  PERMISSION_READ muss gesetzt sein.

- PERMISSION_DELETE

  Annotation kann gelöscht werden, hierzu muss aber auch

  PERMISSION_READ gesetzt sein.

Zur Überprüfung von Berechtigungen bietet die Klasse AnnotationPermission folgende Methoden:

- canWrite(Annotation anno)

  true = Annotation kann gespeichert werden

  false = kein Speichern möglich

- canRead(Annotation anno)

  true = Annotation wird im Viewer angezeigt.

  false = wird nicht im Viewer angezeigt.

- canChange(Annotation anno)

  true = Änderungen an Annotation möglich (Position, Grösse, Eigenschaften)

  false = Keine Änderungen an Annotation möglich, keine Selektierung und kein Ändern der Eigenschaften möglich.

3 com.levigo.jadice.annotation.AnnotationPermission

- canDelete(Annotation anno)

  true = Annotation kann gelöscht werden

  false = Annotation kann nicht gelöscht werden.

Die Berechtigungen können auch kombiniert werden, z.B.:

PERMISSION_READ + PERMISSION_WRITE + PERMISSION_CHANGE + PERMISSION_DELETE

Die obige Berechtigung entspricht der Standardberechtigung einer neu angelegten Annotation.

## 4.1. Berechtigung einer Annotation setzen

Nach dem vollständigem Laden der Annotationen kann man Änderungen an den Berechtigungen vornehmen. Hierzu müssen die Annotationen aus dem AnnotationPageSegment geholt werden (siehe auch Kapitel7).

```
Document document = myDocument;
// Für jede Seite...
for (int i = 0; i < document.getPageCount(); i++) {
  //...suche das entsprechende AnnotationPageSegment.
  AnnotationPageSegment aps = (AnnotationPageSegment)
    document.getPage(i).getPageSegment(document.getLayer(
    AnnotationPageSegment.DEFAULT_LAYER_NAME));
  //Falls es vorhanden ist,...
  if (aps != null) {
    //...iteriere über jede Annotation...
    for (Iterator iter = aps.getAnnotations().iterator();
      iter.hasNext();) {
      Annotation annotation = (Annotation)iter.next();
      //... und ändere die Berechtigung.
      int  permission =
        AnnotationPermission.PERMISSION_NONE;
      annotation.setPermission(permission);
    }
  }
}
```

*Code Beispiel 1 – Annotationen aus AnnotationPageSement holen und Berechtigung ändern.*

## 4.2. Berechtigung einer Annotation setzen während des Ladevorgangs

Annotationsberechtigungen können auch während des Ladevorgangs gesetzt werden. Hierzu gibt es in der AnnotationInit.properties Datei einen Parameter zur Definition einer Klasse, die das Setzen der Berechtigungen ausführt.

VisualInfo / ImagePlus Parameter:

**vi.annotation.permission-apply-class=MyClass**

FileNet Parameter:

**`fn.annotation.permission-apply-class=MyClass`**

Diese Klasse muss vom Integrator realisiert werden. Hierzu muss die Klasse das Interface Annotation PermissionEstablisher[4] implementieren.

Nach dem Laden der Annotationen wird die Methode **`applyPermission(Collection)`** aufgerufen. In der Collection werden alle geladenen Annotationen übergeben, die Berechtigung kann dann entsprechend gesetzt werden.

```
public class MyClass implements
  AnnotationPermissionEstablisher {
  /* (non-Javadoc)
   * @see com.levigo.jadice.annotation.
   * AnnotationPermissionApply#setPermission
   * (java.util.Collection)
   */
  public void applyPermission(Collection annotations) {
    for (Iterator iter = annotations.iterator();
      iter.hasNext(); ) {
      Annotation anno = (Annotation) iter.next();
      // Berechtigung Lesen, Ändern und Schreiben setzen
      int permission =
        AnnotationPermission.PERMISSION_CHANGE
        + AnnotationPermission.PERMISSION_READ
        + AnnotationPermission.PERMISSION_WRITE;
      anno.setPermission(permission);
    }
  }
}
```

*Code Beispiel 2 / Implementierung der Schnittstelle AnnotationPermissionEstablisher*

4 com.levigo.jadice.annotation.AnnotationPermissionEstablisher

# 5. Laden

Im folgenden Abschnitt wird anhand eines Beispiels aufgezeigt, wie Annotationen geladen werden können. Der Übersicht halber wird an dieser Stelle nicht näher auf den Ladevorgang des eigentlichen Bilddokuments eingegangen, sondern ausschließlich auf das Laden von Annotationen.

Der Loader[5] ist die zentrale Klasse des jadice viewer für alle Ladevorgänge, hier im Beispiel zum Laden von Annotationen. Eine detailliertere Beschreibung des Loaders und seiner Anwendung befindet sich in der jadice viewer Integrations-Dokumentation oder der jadice API Referenz.

Zunächst wird eine Instanz des Loaders erzeugt. Wird kein bereits vorhandenes Document[6] dem Loader übergeben, erzeugt dieser ein neues Dokument, welches während des Ladevorgangs „befüllt" wird und dem Viewer zur Anzeige übergeben werden kann.

Im nächsten Schritt wird versucht zu der gegebenen Bild-Datei „file2Load" eine korrespondierende Annotations-Datei zu finden. Annotations-Dateien führen üblicherweise den gleichen Namen wie das Bilddokument und haben als Suffix „.T_L" für VisualInfo bzw. ImagePlus Annotationen oder „.xml" für FileNet Annotationen. Existiert eine solche Datei, wird diese zum Laden der Annotationen verwendet. An dieser Stelle sei darauf hingewiesen, dass Annotations-Daten i.d.R. nicht als Datei, sondern als Stream aus einem Archiv oder ähnlichem vorliegen.

Format-Informationen beschreiben das Format, in dem ein Dokument vorliegt. Gibt man dem Loader in der „loadDocument" Methode keine Format-Information an, versucht der Loader das Format selbst zu bestimmen. Da ImagePlus Annotationen MO:DCA Strukturen sind, ist angeraten, Annotationen immer mit ImagePlusAnnotationFormatInfo[7] zu laden. Anderenfalls kann eine Verwechslung mit MO:DCA nicht ausgeschlossen werden. In einem solchen Fall würden Annotationen als eigenständiges Dokument geladen, würden nicht in dem bekannten Erscheinungsbild dargestellt werden und wären auch nicht vom Benutzer änderbar.



Hinweis:

Das Laden und Speichern von Annotationen muss explizit durchgeführt werden.

Ausnahme:

Eine Hilfsklasse des jadice Pakets ist der FileOpener[8], der Ladevorgänge von Dokumenten und zugehörigen Annotationen automatisch vornimmt.

## 5.1. Laden von VisualInfo / ImagePlus Annotationen

Das folgende Beispiel zeigt, wie ImagePlus kompatible Annotationen geladen werden können.

5 com.levigo.jadice.docs.resource.Loader
6 com.levigo.jadice.docs.Document
7 com.levigo.jadice.formats.annoiplus.ImagePlusAnnotationFormatInfo
8 com.levigo.jadice.util.FileOpener

```
File file2Load = new File("MeinBild.tif");
Loader loader = new Loader();
// Dokument laden...
int lastDot = file2Load.lastIndexOf(".");
if (lastDot > 0) {
  // versuche Annotations Datei zu finden
  String annoFileName = file2Load.substring(0, lastDot);
  // Default Annotation Extension: „.T_L"
  File annoFile = new File(annoFileName + ".T_L");
  // lade Annotationen, wenn Datei existiert
  if (annoFile.exists())
    loader.loadDocument(new FileInputStream(annoFile),
      new ImagePlusAnnotationFormatInfo(), 0);
}
```

*Code Beispiel 3- VisualInfo / ImagePlus Annotationen laden*

## 5.2. Laden von FileNet Annotationen

Im folgenden Beispiel wird aufgezeigt, wie FileNet kompatible Annotationen geladen werden können.

Hinweis:

Die Methode FileNetAnnotationFile.convertToUTF8(InputStream) dient als Workaround, da die zur Zeit der Implementierung aktuelle FileNet Image Services Resource Adapter (Version 3.0a) Schnittstelle einen NICHT konformen UTF-8 Datenstrom liefert. Bereits korrekte UTF-8 XML Daten sollten nicht mit dieser Methode bearbeitet werden, anderenfalls kann nicht für eine korrekte Datenverarbeitung garantiert werden.

```
File file2Load = new File("MeinBild.tif");
Loader loader = new Loader();
// Dokument laden...
int lastDot = file2Load.lastIndexOf(".");
if (lastDot > 0) {
  // versuche Annotations Datei zu finden
  String annoFileName = file2Load.substring(0, lastDot);
  // Default Annotation Extension: „.xml"
  File annoFile = new File(annoFileName + ".xml");
  // lade Annotationen, wenn Datei existiert
  if (annoFile.exists())
    loader.loadDocument(
      FileNetAnnotationFile.convertToUTF8(new
      FileInputStream(annoFile))
      new FileNetAnnotationFormatInfo(), 0);
}
```

*Code Beispiel 4- FileNet Annotationen laden*

## 5.3. Laden von FileNet P8 Annotationen

Im folgenden Beispiel wird aufgezeigt, wie FileNet P8 Annotationen geladen werden können. Annotationen werden einzeln vom Archiv geladen. Jede Annotation hat einen eigenen Datenstrom. Mit der FileNetP8AnnotationInputXMLParser-Klasse werden die einzelnen

Annotationsdatenströme zusammengefasst und in einen Datenstrom geschrieben, der dann mit der FileNetP8AnnotationFile-Klasse geladen wird.

```
// Input-Handler erstellen
FileNetP8AnnotationInputXMLParser input = new
FileNetP8AnnotationInputXMLParser();

// alle Annotationen zusammenfassen
input.addAnnotationXMLStream(>Annotation 1 Datenstrom<);
input.addAnnotationXMLStream(>Annotation 2 Datenstrom<);
input.addAnnotationXMLStream(>Annotation 3 Datenstrom<);
input.addAnnotationXMLStream(>Annotation x Datenstrom<);

// Datenstrom erstellen
ByteArrayOutputStream bos = new ByteArrayOutputStream();

// für ein- und mehrseitiges Dokument (ein Datenstrom)
alle Annotationen
// schreiben
input.write(bos);

// für ein zusammengesetztes Dokument (mehrere
// Datenströme)
// hier werden nur die Annotationen für die gewünschte
// Seite gespeichert
input.write(bos, >Seitenindex des Dokuments<);

// Daten holen
byte[] annotationData = bos.toByteArray();
bos.close();

// Datenstrom für den Ladevorgang erstellen
ByteArrayInputStream bis = new
  ByteArrayInputStream(annotationData);

// Annotationen laden
FileNetP8AnnotationFile file = new
  FileNetP8AnnotationFile(>jadice Dokument<);
file.load(bis, document
  .getLayer(AnnotationPageSegment.DEFAULT_LAYER_NAME), 0,
  null);
// alternativ: Laden über die Loader-Klasse
Loader loader = new Loader();
loader.loadDocument(bis, new
  FileNetP8AnnotationFormatInfo(), 0);
```

*Code Beispiel 5- FileNet P8 Annotationen laden*

# 6. Speichern

Ähnlich wie Format-Informationen Ladevorgänge in bestimmte Formate unterstützen, gibt es Klassen, die Speichervorgänge von bestimmten Formaten (*FormatName*File) unterstützen. Die Benennung dieser Klassen folgt einer vorgegebenen Namenskonvention.

**Beispiel:**

*FormatName*FormatInfo -> TIFFFormatInfo[9]

*FormatName*File -> TIFFFile[10]

Für ImagePlus kompatible Annotationen heißt diese Klasse dementsprechend ImagePlusAnnotationFile[11], für FileNet kompatible Annotationen heißt diese Klasse FileNetAnnotationFile[12].

Das Speichern wird hier am Beispiel der ImagePlusAnnotationFile Klasse beschrieben.

Zunächst wird eine Instanz von ImagePlusAnnotationFile erzeugt. Um einen Zugriff auf die zu speichernden Annotationen zu erhalten, wird dem ImagePlusAnnotationFile im Konstruktor das Dokument übergeben, dessen Annotationen zu speichern sind.

Ebenso wie der Loader verschiedene Lademethoden zur Verfügung stellt, erlaubt auch jede FormatNameFile-Instanz qualifiziert (z.B. nur bestimmte Seiten o.ä.) zu speichern. Genauere Informationen entnehmen Sie bitte der jadice viewer Integrations-Dokumentation.

In Code Beispiel 5 wurde die einfachste Methode gewählt, um alle Annotationen eines Dokuments zu speichern. Man übergibt der ImagePlusAnnotationFile Instanz einfach einen OutputStream, in dem die Annotations-Informationen abgelegt werden sollen.

In Code Beispiel 7 wird die Speicherung von FileNet Annotationen gezeigt. Hier gibt es einige Punkte zu beachten.

Hinweis:

Das Laden und Speichern von Annotationen muss explizit durchgeführt werden.

Ausnahme:

Eine Hilfsklasse des jadice Pakets ist der FileOpener[13], der Ladevorgänge von Dokumenten und zugehörigen Annotationen automatisch vornimmt.

In den folgenden Abschnitten wird anhand eines einfachen Beispiels aufgezeigt, wie ImagePlus und FileNet kompatible Annotationen in eine Datei gespeichert werden können.

9 com.levigo.jadice.formats.tiff.TIFFFormatInfo
10 com.levigo.jadice.formats.tiff.TIFFFile
11 com.levigo.jadice.formats.annoiplus.ImagePlusAnnotationFile
12 com.levigo.jadice.formats.annofilenet.FileNetAnnotationFile
13 com.levigo.jadice.util.FileOpener

## 6.1. Speichern von VisualInfo / ImagePlus Annotationen

Hier gibt es die Möglichkeit zusätzliche Informationen zu den Annotationen mitzuspeichern.

Zu beachten ist, dass Annotationen mit zusätzlich gespeicherten Infos in jedem VisualInfo Client angezeigt werden können. Sobald diese Annotationen jedoch mit einem VisualInfo Client wieder gespeichert werden, gehen die zusätzlichen Infos verloren.

Um das Speichern der Infos zu aktivieren, muss in der AnnotationInit.properties Datei folgender Parameter entsprechend gesetzt werden:

**vi.annotation.save-additional-info=<Wert>**

Wert: true = Infos werden gespeichert.

false = Infos werden nicht gespeichert

Die Standardeinstellung ist *false*.

```
// ähnlich wie beim Laden:
ImagePlusAnnotationFormatInfo // ->
ImagePlusAnnotationFile nehmen
ImagePlusAnnotationFile annoFile = new
  ImagePlusAnnotationFile(myViewer.getDocument());
try {
  annoFile.save(new FileOutputStream("MeinBild.T_L"));
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

*Code Beispiel 6 – Annotationen speichern*

## 6.2. Speichern von FileNet Annotationen

Beim Speichern von FileNet Annotationen müssen zusätzliche FileNet Informationen angegeben werden, damit die Annotationen korrekt ins FileNet Archiv gespeichert werden können. Diese Angaben werden mit jedem Speichervorgang durch FileNet verändert. Um einen konsistenten Zustand zwischen Archiv und angezeigten Annotationen zu bewahren, muss das Speichern von FileNet Annotationen in drei Schritten erfolgen:

- Speichern der Annotationen im Archiv.
- Löschen der Annotationen im jadice Dokument, damit diese später nicht doppelt im Dokument erscheinen.
- Erneutes Laden der Annotationen von FileNet mit aktualisierten FileNet Informationen (z.B. Zeitstempel, Annotationsidentifikationsnummer).

Folglich müssen nach dem Speichern alle vorhandenen Annotationen aus dem AnnotationPageSegment entfernt werden (siehe Codebeispiel 7).

```
// Annotation Layer holen
DocumentLayer annoLayer = document.getLayer
  (AnnotationPageSegment.DEFAULT_LAYER_NAME);
// Alle Seiten bearbeiten
for (Iterator it = document.getPages().iterator();
  it.hasNext();) {
  Page aPage = (Page) it.next();
  // Annotationpagesegment für aktuelle Seiten holen
  AnnotationPageSegment aps = (AnnotationPageSegment)
    aPage.getPageSegment(annoLayer);
  if (aps != null) {
    // Alle Annos deselektieren
    aps.deselectAllAnnotations(aps.getAnnotations());
    // Annos entfernen
    aps.getAnnotations().clear();
    aps.getDeletedAnnotations().clear();
  }
}
```

*Code Beispiel 7- Annotationen aus AnnotationPageSegment entfernen*

Nach erfolgreichem Löschen der Annotationen im AnnotationPageSegment, müssen nun die von FileNet aktualisierten Annotationen erneut geladen werden. Der Ladevorgang von Annotationen aus FileNet ist im Kapitel 5.2 beschrieben.

Zum korrekten Speichern von Annotationen ins FileNet Archiv müssen zusätzliche FileNet spezifische Informationen angegeben werden.

Folgende Informationen werden benötigt, die zur Erstellung eines FileNetAnnotationFile Objekts in dessen Konstruktor übergeben werden:

FileNet Archiv Definition:

Die FileNet Archiv Definition setzt sich aus folgenden Parametern zusammen:

library = FileNet Library Name (Default: „DefaultIMS")

domain = FileNet Domain Name (Default: „Imaging")

organisation = FileNet Organisation Name (Default: „FileNet")

FileNet Dokument ID:

ID des Dokuments, auf das die Annotationen gespeichert werden sollen:

docId = FileNet Dokument ID

FileNet Client Zugriffsrechte:

Zugriffsberechtigung des Clients:

clientPermission = Client Zugriffsberechtigung, möglich sind hier „none", „change", „admin".

FileNet Zugriffsberechtigungen:

Zugriffsberechtigungen des Clients (Benutzer oder Gruppe):

permissionTypeAppend,

permissionTypeRead,

permissionTypeWrite =

Zugriffsberechtigung, möglich sind hier „user", „group".

FileNet Benutzerberechtigungen:

Benutzerberechtigungen des Clients (Benutzer oder Gruppe):

permissionNameAppend,

permissionNameRead,

permissionNameWrite =

Benutzerberechtigung, hier muss ein gültiger FileNet Benutzer / FileNet Benutzergruppe angegeben werden.

Standardbenutzer sind „(ANYONE)" und „(NONE)".

Steuern der Berechtigung:

Wenn das useDefaultPermission Flag auf „false" gesetzt wird, dann werden bereits existierende Annotationen mit der vorhandenen (=zuvor geladenen) Berechtigung abgelegt. Neu angelegte Annotationen werden mit der im Konstruktor definierten Berechtigung gespeichert.

Wenn das Flag auf „true" gesetzt wird, dann werden alle Annotationen mit der im Konstruktor definierten Berechtigung gespeichert.

```
// FileNet Archiv Definition
String library = "DefaultIMS";
String domain = "Imaging";
String organisation = "FileNet";
// FileNet Dokument ID
String docId = "100000";
// Client Zugriffsrechte
String clientPermission = "change";
// Zugriffsberechtigungen
String permissionTypeAppend = "user";
String permissionTypeRead = "user";
String permissionTypeWrite = "user";
// Benutzerberechtigungen
String permissionNameAppend = "(ANYONE)";
String permissionNameRead = "(ANYONE)";
String permissionNameWrite = "(ANYONE)";
// Flag zum Steuern der Zugriffsrechte.
// false = bei bereits existierenden Annotationen wird das
// vorhandene Zugriffsrecht benutzt, neu angelegte
// Annotationen bekommen das oben definierte Zugriffsrecht
// zugewiesen.
// true = alle Annotationen bekommen das oben
definierte // Zugriffsrecht zugewiesen
boolean useDefaultPermission = false;
FileNetAnnotationFile file = new FileNetAnnotationFile(
  myViewer.getDocument(),
  library,
  domain,
  organisation,
  docId,
  clientPermission,
  permissionTypeAppend,
  permissionTypeRead,
  permissionTypeWrite,
  permissionNameAppend,
  permissionNameRead,
  permissionNameWrite,
  useDefaultPermission);
```

```
try {
  annoFile.save(new FileOutputStream("MeinBild.xml"));
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

*Code Beispiel 8 – Annotationen speichern*

## 6.3. Speichern von FileNet P8 Annotationen

Beim Speichern von FileNet P8 Annotationen muss folgende Reihenfolge beachtet werden, um einen konsistenten Zustand zwischen Archiv und angezeigten Annotationen zu bewahren:

- Speichern der Annotationen im Archiv.
- Löschen der Annotationen im jadice Dokument, damit diese später nicht doppelt im Dokument erscheinen.
- Erneutes Laden der Annotationen vom Archiv

Nach erfolgreichem Löschen der Annotationen im AnnotationPageSegment, müssen nun die von FileNet aktualisierten Annotationen erneut geladen werden. Der Ladevorgang ist im Kapitel 5.3 beschrieben.

Es gibt 4 Zustände für Annotationen, die den weiteren Verlauf des Speichervorgangs bestimmen:

- **Unverändert** (Unmodified)

  Unveränderte Annotationen müssen nicht im Archiv gespeichert werden.

- **Verändert** (Modified)

  Veränderte Annotationen müssen im Archiv gespeichert werden, die Zeitangabe wird aktualisiert (betrifft Attribut F_MODIFYDATE).

- **Hinzugefügt** (Added)

  Hinzugefügte Annotationen müssen im Archiv gespeichert werden, die Zeitangabe wird gesetzt (Attribute F_ENTRYDATE und F_MODIFYDATE), folgende Attribute müssen zusätzlich angegeben werden:

  - ID (Attribute F_ID und F_ANNOTATEDID)

    Neue Annotation-ID, muss vom Archiv geholt werden.

  - Seitennummer (Attribut F_PAGENUMBER)

    Bei zusammengesetzten Dokumenten muss noch die entsprechende Seitennummer gesetzt werden, Standardwert ist 1.

- **Gelöscht** (Deleted)

  Gelöschte Annotationen müssen aus dem Archiv entfernt werden. Die Annotationen werden anhand der ID geladen und dann gelöscht.

Mit der FileNetP8AnnotationOutputXMLParser-Klasse kann man auf die verschiedenen Zustände der Annotationen zugreifen, alternativ ist dies auch über die FileNetP8AnnotationFile-Klasse möglich.

```
// FileNetP8AnnotationFile erstellen
FileNetP8AnnotationFile file = new
FileNetP8AnnotationFile(>jadice Dokument<);

// Annotationen speichern
ByteArrayOutputStream os = new ByteArrayOutputStream();
file.save(os);
byte[] annotationData = os.toByteArray();
os.close();

// Output-Handler erstellen
ByteArrayInputStream is = new
ByteArrayInputStream(annotationData);
FileNetP8AnnotationOutputXMLParser output =
    new FileNetP8AnnotationOutputXMLParser(is);

org.w3c.dom.Document[] annotations = null;

//#########################################################
// Unveränderte Annotationen
//#########################################################

annotations = output.getUnmodifiedAnnotations();
// alternativ:
annotations = file.getUnmodifiedAnnotations(null);

// alle Annotationen bearbeiten
for (int i = 0; i < annotations.length; i++) {
  org.w3c.dom.Document anno = annotations[i];
  // kein Speichern nötig, hier nur Debug-Ausgabe
  // Annotation-ID holen
  String id =
    FileNetP8AnnotationXMLUtils.getAnnotationID(anno);
  System.out.println("Anno " + id + " unverändert !");
}

//#########################################################
// Veränderte Annotationen
//#########################################################

annotations = output.getModifiedAnnotations();
// alternativ:
annotations = file.getModifiedAnnotations(null);

// alle Annotationen bearbeiten
for (int i = 0; i < annotations.length; i++) {
  org.w3c.dom.Document anno = annotations[i];
  ByteArrayOutputStream bos = new ByteArrayOutputStream();
  FileNetP8AnnotationXMLUtils.write(bos, anno);
  byte[] data = bos.toByteArray();
  bos.close();
  // Annotation-ID holen
  String id =
    FileNetP8AnnotationXMLUtils.getAnnotationID(anno);
  // neuen Datenstrom erstellen für das Speichern ins
  // Archiv
  ByteArrayInputStream bis = new
    ByteArrayInputStream(data);

  // hier muss die Annotation aus dem Archiv anhand der ID
```

```
  // geladen und der Datenstrom über das Annotationsobjekt
  // geschrieben werden
  // >>>>>> Archivzugriff

  System.out.println("Anno " + id + " aktualisiert !");
}

//#########################################################
// Hinzugefügte Annotationen
//#########################################################

annotations = output.getAddedAnnotations();
// alternativ:
annotations = file.getAddedAnnotations(null);

// alle Annotationen bearbeiten
for (int i = 0; i < annotations.length; i++) {
  org.w3c.dom.Document anno = annotations[i];
  ByteArrayOutputStream bos = new ByteArrayOutputStream();
  FileNetP8AnnotationXMLUtils.write(bos, anno);
  byte[] data = bos.toByteArray();
  bos.close();

  // hier muss ein neues Annotationsobjekt im Archiv
  // erzeugt und die ID geholt werden
  // >>>>>> Archivzugriff

  String id = "Neue ID";
  // neue ID setzen
  FileNetP8AnnotationXMLUtils.setAnnotationID(anno, id);
  // evtl. Seitennummer für Multi-Dokumente setzen
  FileNetP8AnnotationXMLUtils.setPageNumber(anno,
    >Seitennummer<);
  // neuen Datenstrom erstellen für das Speichern ins
  // Archiv
  ByteArrayInputStream bis = new
    ByteArrayInputStream(data);

  // hier muss der Datenstrom über das Annotationsobjekt
  // geschrieben werden
  // >>>>>> Archivzugriff

  System.out.println("Anno " + id + " hinzugefügt !");
}

//#########################################################
// Gelöschte Annotationen
//#########################################################

annotations = output.getDeletedAnnotations();
// alternativ:
annotations = file.getDeletedAnnotations(null);

// alle Annotationen bearbeiten
for (int i = 0; i < annotations.length; i++) {
  org.w3c.dom.Document anno = annotations[i];
  // ID der Annotation holen
  String id =
    FileNetP8AnnotationXMLUtils.getAnnotationID(anno);

  // hier muss die Annotation aus dem Archiv anhand der ID
```

```
  // geladen und dann gelöscht werden
  // >>>>>> Archivzugriff

  System.out.println("Anno " + id + " gelöscht !");
}

// alle Annotationen vom Dokument entfernen
file.removeAllAnnotationsFromDocument(null);
```

*Code Beispiel 9 – Annotationen speichern*

# 7. Zugriff

In diesem Abschnitt wird erklärt, wie Integratoren direkten Zugriff auf seitenzugehörige Annotationen erhalten.

Um den Zugriff verständlich zu machen, zunächst ein paar Vorbemerkungen zum Dokument- und Seitenmodell des jadice viewer.

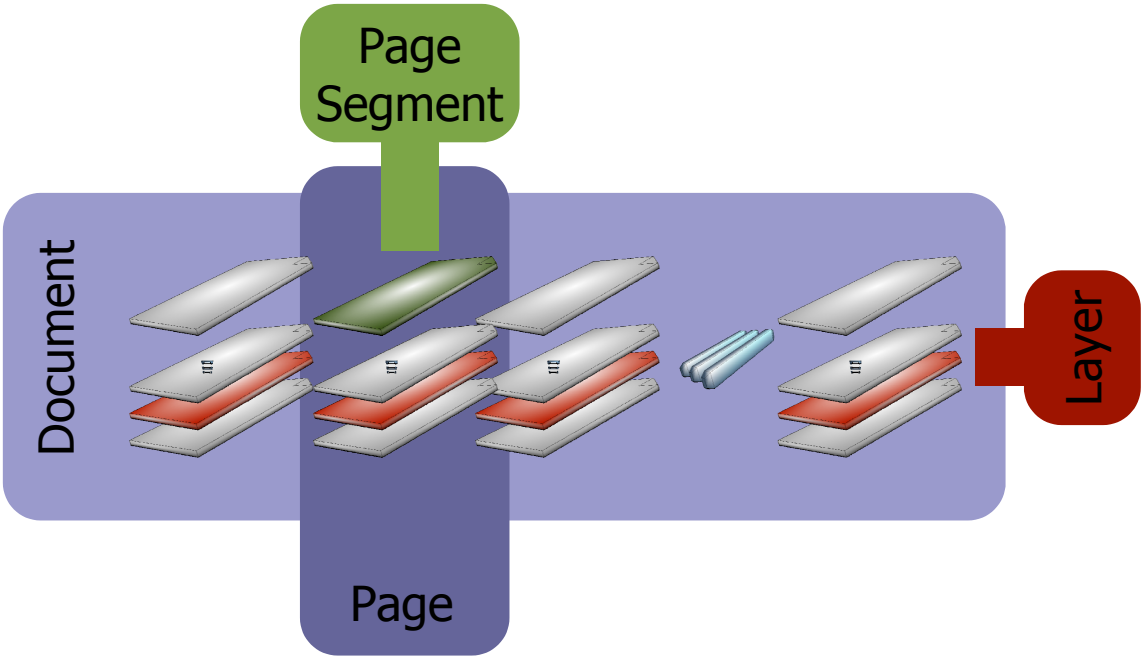


*Abbildung 2 Dokumentenmodell*

Ein Dokument besteht aus einer oder mehreren Seiten, die in der Applikation als eine Einheit betrachtet werden. Eine Seite kann aus mehreren Schichten bestehen, die jedoch als eine Einheit visualisiert werden. Diese Schichten nennt man Darstellungsebenen oder Layer. In diesen Ebenen befinden sich Seitensegmente, die Daten aus verschiedenen Datenquellen repräsentieren. Beispiel: eine Ebene ist ein Briefkopf, eine weitere ein Briefinhalt als Text.

Eine detailliertere Beschreibung des Dokument-Modells entnehmen Sie bitte der jadice viewer Integrations-Dokumentation oder der jadice API Referenz.

Da Annotationen nicht Teil eines Dokuments sind, sondern eine zusätzliche Informationsquelle, werden Annotationen in einem eigenen Seitensegment, genauer dem AnnotationPageSegment[14], verwaltet und dargestellt.

14 com.levigo.jadice.annotation.AnnotationPageSegment

## 7.1. ... auf das AnnotationPageSegment

Wie im vorigen Abschnitt erläutert, ist das AnnotationPageSegment die zentrale Klasse, um direkte Referenzen auf die aktuell zu einer Seite vorhandenen Annotationen zu erhalten.

Es folgt ein Code Beispiel, in dem Schritt für Schritt der Zugriff auf das AnnotationPageSegment einer Seite beschrieben ist.

```
public AnnotationPageSegment getAnnotationPageSegment(Page
  page) {
  AnnotationPageSegment aps = null;

  // suche die Darstellungsebene, in der die Annotationen
  // liegen
  Document doc = page.getParentDocument();
  DocumentLayer layer = doc.getLayer(
    AnnotationPageSegment.DEFAULT_LAYER_NAME);

  // ist die Ebene nicht vorhanden, existiert kein
  // AnnotationPageSegment
  if (layer != null){
    // erhalte das Seiten Segment der Ebene
    aps = (AnnotationPageSegment)
      page.getPageSegment(layer);
  }
  return aps;
}
```

*Code Beispiel 10 / Referenz auf AnnotationPageSegment*

Um eine Referenz auf ein bestimmtes Segment einer Seite zu erhalten, muss zunächst ermittelt werden, in welcher Darstellungsebene sich das gesuchte Seitensegment befindet. Im Allgemeinen existiert nur eine Darstellungsebene für Annotationen.

Eine Referenz auf den AnnotationsDefaultLayer erhält man über die Layer Zugriffsmethoden des Dokuments.

Ist kein solcher Layer vorhanden, hat das gesamte Dokument keine Annotationen und damit auch kein AnnotationPageSegment.

Andernfalls kann man nun mittels des Annotations-Layers die Seite nach dem Segment in diesem Layer befragen. Dieses Segment ist das gesuchte AnnotationsPageSegment.

## 7.2. ... auf Annotationen einer Seite

Eine Instanz der Klasse AnnotationPageSegment bietet über einfache „getter"-Methoden Zugriff auf alle Annotationen einer Seite bzw. auf alle selektierten Annotationen einer Seite.

```
// Zugriffsmethode aus Code Beispiel 10
AnnotationPageSegment aps =
  getAnnotationPageSegment(aPage);

  // alle Annotationen einer Seite
```

```
Collection allAnnotations = aps.getAnnotations();
// alle selektierten Annotationen einer Seite
Collection allSelectedAnnotations =
  aps.getSelectedAnnotations();
```

*Code Beispiel 11 / Referenz auf alle bzw. alle selektierten Annotationen einer Seite*

# 8. Verändern von Annotationen

Zum besseren Verständnis der folgenden Kapitel, wird an dieser Stelle kurz auf den Zusammenhang zwischen dem Dokument- und dem Device-Koordinatensystem eingegangen.

Wie in Abbildung 2 (jadice viewer Dokumenten-Modell) ersichtlich (vergeichen Sie dazu auch die jadice viewer Integrations-Dokumentation), können die in Dokumenten beinhalteten Seiten ihren Inhalt aus mehreren Datenströmen bzw. Datenformaten beziehen, aber auch die Seiten selbst können sich in Ihrer Darstellung zusammensetzen aus Daten verschiedener Datenströme. Da diese Bilddaten verschiedenen Formats und Auflösung sein können, unterhält der Viewer intern ein Dokument-Koordinaten-System und transformiert nur zur Darstellung in die jeweiligen Device-Koordinaten. Dementsprechend halten Seitensegmente, wie auch das AnnotationPageSegment, ihre Daten in Dokumenten-Koordinaten.

Folglich sind Positions- und Größenangaben, wie auch Änderungen an Annotationen in Dokument-Koordinaten zu verstehen. Zur einfachen Umrechnung von Device- zu Dokument- und Dokument- zu Device-Koordinaten bietet die Klasse RenderContext[15] entsprechende affine Transformationen. Detailliertere Angaben zur Klasse RenderContext befinden sich in Kapitel 9 oder der Integrations-Dokumentation des jadice viewer. Beispiele zur Anwendung der entsprechenden affinen Transformationen werden in den folgenden Kapiteln geliefert.

## 8.1. Hinzufügen und Löschen

Die Verbindung zwischen Annotationen und der Seite eines Dokuments ist das AnnotationPageSegment. Hinzufügen und Löschen geschieht demzufolge unter Zuhilfenahme des AnnotationPage-Segments.

```
// Zugriffsmethode aus Code Beispiel 10
AnnotationPageSegment aps =
  getAnnotationPageSegment(aPage);

if (aps != null) {
  // lösche eine bestimmte Annotation
  aps.deleteAnnotation(anAnnotation);

  // lösche alle ausgewählten Annotationen
  aps.deleteSelectedAnnotations();
}
```

*Code Beispiel 12 / Löschen von Annotationen*

Wenn kein AnnotationPageSegment zu einer gegebenen Seite existiert, besitzt diese Seite keine Annotationen. Damit ist ein Löschvorgang obsolet. Andernfalls bietet das AnnotationPageSegment zwei Methoden zum Löschen. Mit der „deleteAnnotation" Methode kann eine bestimmte Annotation qualifiziert entfernt werden; mit der „deleteSelectedAnnotations" werden alle ausgewählten Annotationen entfernt.

15 com.levigo.jadice.docs.RenderContext

Soll nun eine Annotation der Seite hinzugefügt werden, muss sichergestellt sein, dass ein AnnotationPageSegment zu dieser Seite vorhanden ist.

Im Code Beispiel 10 wurde erläutert, wie man Zugriff auf das AnnotationPageSegment erhält. Wenn nun aber das Dokument noch keine Annotationen enthält, kann es vorkommen, dass das Dokument noch keinen Annotations-Layer enthält bzw. die Seite noch kein AnnotationPageSegment. Im folgenden Beispiel wurde die Zugriffsmethode auf das AnnotationPageSegment aus Code Beispiel 10 so erweitert, dass nun ein entsprechender Layer bzw. AnnotationPage-Segment angelegt wird.

```
public AnnotationPageSegment getAnnotationPageSegment(Page
  page) {
  AnnotationPageSegment aps = null;

  // suche die Darstellungsebene, in der die Annotationen
  // liegen
  Document doc = page.getParentDocument();
  DocumentLayer layer = doc.getLayer(
    AnnotationPageSegment.DEFAULT_LAYER_NAME);

  // ist die Ebene nicht vorhanden, lege sie an
  if (layer == null)
    layer = doc.addLayer(
      AnnotationPageSegment.DEFAULT_LAYER_NAME,
      DocumentLayer.TOP);

  // erhalte das Seitensegment der Ebene
  aps = (AnnotationPageSegment)
    page.getPageSegment(layer);

  // ist das Seitensegment nicht vorhanden, lege es an
  if (null == aps) {
    aps = new AnnotationPageSegment(page);
    page.addPageSegment(aps, layer);
  }

  return aps;
}
```

*Code Beispiel 13- Referenz auf AnnotationPageSegment, erweitert*

Mit der Zugriffsmethode aus Code Beispiel 11 können Sie nun Annotationen hinzufügen und sicher sein, dass ein entsprechender Layer im Dokument vorhanden ist und die Seite ein AnnotationPageSegment besitzt.

Das eigentliche Hinzufügen erfolgt nun einfach mittels der Methode „addAnnotation" des AnnotationPageSegments.

Hinweis:
Das Hinzufügen und Entfernen von Annotationen löst ein „PageModified" Event bei allen registrierten DocumentListener[16] des Dokuments aus.

16 com.levigo.jadice.docs.DocumentListener

## 8.2. Größe und Position

Alle Annotationen stammen von der Basis-Klasse Annotation[17] ab. Diese Klasse stellt für Größen- und Positionsveränderungen jeweils eine entsprechende Methode zur Verfügung. Somit können Sie, unabhängig vom Typ der Annotation, diese Methoden nutzen. *Die einzige Ausnahme bildet z.Zt. die StampAnnotation[18]: Diese Annotation bestimmt aufgrund ihrer Eigenschaften ihre Größe selbstständig.*

Die Methoden lauten im Detail:

- **Annotation.setLocation(Point)** - zur Veränderung der Position
- **Annotation.setSize(Dimension)** - zur Veränderung der Größe.

Hinweis:

Die Ursprungsposition der Pfeil-Annotation bezieht sich immer auf die Pfeilspitze.

Hinweis:

Alle Größen- und Positionsangaben sind in Dokument-Koordinaten zu verstehen.

Hinweis:

Integratoren sollten nach Veränderungen an Annotationen den Viewer neu rendern lassen. Beispiel: myViewer.repaint();

Hinweis:

Bitte beachten Sie: Direkte Änderungen an Annotationen lösen KEIN „PageModified" Event bei registrierten DocumentListener[19] des Dokuments aus!

## 8.3. Sonstige Änderungen

Entsprechend ihrem Typ haben die einzelnen Annotationen verschiedene Ausprägungen, z.B Vordergrundfarbe, Hintergrundfarbe, Text etc. Detaillierte Informationen zu jedem Annotationstyp finden Sie in der jadice viewer API.

Integratoren sind frei Attribute von Annotationen zu ändern. Eventuelle Kompatibilitäts-Einschränkungen sind jedoch zu beachten.

Hinweis:

Integratoren sollten nach Veränderungen an Annotationen den Viewer neu rendern lassen. Beispiel: myViewer.repaint();

Diese „repaint"-Aufforderung wird in einer zukünftigen Version des jadice viewer automatisiert stattfinden und nicht mehr manuell gestartet werden müssen.

Hinweis:

Bitte beachten Sie: Veränderungen direkt an Annotationen lösen KEIN „PageModified" Event bei allen registrierten DocumentListener[20] des Dokuments aus!

17 com.lavigo.jadice.annotation.Annotation
18 com.levigo.jadice.annotation.StampAnnotation
19 com.levigo.jadice.docs.DocumentListener
20 com.levigo.jadice.docs.DocumentListener

## 8.4. Ereignisse von Annotationsänderungen

Um auf Änderungen von Annotationen reagieren zu können, besteht die Möglichkeit, eine Klasse, die das AnnotationListener[21]-Interface implementiert, über die statische addAnnotationListener-Methode der Klasse AnnotationEventCaster[22] zu setzen.

Bei jeder Änderung der Annotationseigenschaften oder -zustände wird die annotationChanged-Methode der Listenerklasse aufgerufen und es wird eine AnnotationEvent-Klasse übergeben, die Informationen über die entsprechende Änderung beinhaltet.

Informationen aus der AnnotationEvent[23]-Klasse:

- Ereignissart
- betroffene Annotationsklasse
- Alter und neuer Wert der Eigenschaft, die geändert wurde

Die Propagierung von Änderungsereignissen kann für Annotationen unterdrückt werden. Das Verhalten wird über die **setDoFireAnnotationEvents**-Methode der Klasse Annotation gesteuert.

21 com.levigo.jadice.annotation.AnnotationListener
22 com.levigo.jadice.annotation.AnnotationEventcaster
23 com.levigo.jadice.annotation.AnnotationEvent

# 9. Annotationen erzeugen

Dieses Kapitel beschäftigt sich damit, Annotationen programmatisch zu erzeugen. Dazu werden anhand eines Beispiels, in dem eine Rechteck-Annotation an einer zufälligen Position mit einer zufälligen Größe erzeugt wird, die einzelnen Schritte erklärt. Des weiteren wird Bezug genommen auf Kapitel 8.1 Hinzufügen und Löschen.

In Code Beispiel 14 wird zunächst ein zufälliges Rechteck erzeugt. Dieses Rechteck bestimmt im Folgenden die Position und Größe der neu zu erzeugenden RectangleAnnotation[24]. Da Größen- und Positionsangaben in Dokument-Koordinaten gesetzt werden sollten, muss das Rechteck in Dokument-Koordinaten transformiert werden.

```
// zufälliges Rechteck erzeugen
Rectangle annoBoundsDev = new Rectangle(
  (int) Math.round(Math.random() * 300),
  (int) Math.round(Math.random() * 300),
  (int) Math.round(Math.random() * 500),
  (int) Math.round(Math.random() * 500));

// transformiere Rechteck in Dokument-Koordinaten
Rectangle annoBoundsDoc =
  getDoc2DevTransformationForAnnos()
  .createTransformedShape(annoBoundsDev).getBounds();

// erzeuge die Annotation
RectangleAnnotation newRectangleAnnotation =
  new RectangleAnnotation(
  annoBoundsDoc.x,annoBoundsDoc.y,
  annoBoundsDoc.width,annoBoundsDoc.height);
// setze die Farbe auf Rot
newRectangleAnnotation.setForegroundColor(Color.red);

// füge die Annotation dem AnnotationPageSegment hinzu
// getAnnotationPageSegment() aus Code Beispiel 13
getAnnotationPageSegment().addAnnotation(
  newRectangleAnnotation);
```

*Code Beispiel 14 / Rechteck Annotation erzeugen*

Die Transformation in Dokument-Koordinaten erfolgt in zwei Schritten. Zunächst wird in der **„createDoc2DevTransformationForAnnos"** Methode eine entsprechende Transformation erzeugt, dann mittels der **„createTransformedShape"**-Methode der Klasse AffineTransform[25] transformiert. Die Funktionsweise der Methode **„createDoc2DevTransformationForAnnos"** wird später in diesem Kapitel näher beleuchtet. Näheres zu Affinen Transformationen befindet sich in der Java 2 Platform API Specification.

Nachdem nun Größe und Position der neuen Annotation bestimmt sind, wird eine neue Instanz einer RectangleAnnotation erzeugt. Dazu wird der einzige öffentliche Konstruktor verwendet, den RectangleAnnotation zur Verfügung

24 com.levigo.jadice.annotation.RectangleAnnotation
25 java.awt.geom.AffineTransform

stellt. Zur besseren Unterscheidung zwischen programmatisch und durch den Viewer erstellten RectangleAnnotations, wurde hier der Annotation eine rote Vordergrund-Farbe gesetzt. Standardmäßig sind Rectangle-Annotations, die durch den Viewer erstellt wurden, gelb.

Im letzten Schritt wird die neu erzeugte Annotation der Seite hinzugefügt. Dazu wurde eine Referenz auf das AnnotationPageSegment unter Zuhilfenahme von Code Beispiel 13 erzeugt. Wie in Kapitel 8.1 beschrieben, wird die Annotation der Seite hinzugefügt.

Offen bleibt die Frage nach der Affinen Transformation. Wie bereits erwähnt, stellen Instanzen der Klasse RenderContext[26] Affine Transformationen zur Verfügung, die eine einfache Umwandlung zwischen Device und Dokument Koordinaten erlauben. Eine Instanz des RenderContexts mit den aktuellen Setzungen von Zoom, Rotation u.ä. ist stets über den Viewer zu erhalten.

Seitensegmente erzeugen während des Darstellungsprozesses keine Images, sie rendern vielmehr ihre Dateninformation unter Berücksichtigung des gesetzten Zoom-und Rotations-Faktors in einen gegebenen Device-Kontext. Demzufolge sind Positions- und Größenangaben von Annotationen stets mit Rotation 0° und Zoom 100 zu verstehen.

In Code Beispiel 15 wird zunächst der aktuelle RenderContext des Viewer geklont. Somit kann, ohne die Dokument-Darstellung des Viewer zu verändern, die benötigte Instanz eines RenderContexts auf Zoom 100 und Rotation 0° gesetzt werden.

```
public AffineTransform getDoc2DevTransformationForAnnos()
  {
  RenderContext rc = myViewer.getRenderContext().clone();
  // Annos werden immer mit Zoom 100 und Rotation 0
  // erstellt.
  // Zoom und Rotation werden erst während des Renderns
  // angewendet
  rc.setRotation(0);
  rc.setZoomFactor(100);

  // getInverseTransform()  -> Device- zu
  // Dokumentkoordinaten Transformation
  return rc.getInverseTransform();
}
```

*Code Beispiel 15 / Affine Transformation*

Die zentralen Methoden des RenderContexts zur Transformation zwischen Dokument- und Device-Koordinaten sind folgende:

† **RenderContext.getAffineTransform()** - liefert eine Transformation von Dokument- zu Device-Koordinaten

† **RenderContext.getInverseTransform()** - liefert eine Transformation von Device- zu Dokument-Koordinaten

Neu erzeugte Annotationen benötigen Positions- und Grössenangaben in Dokumentkoordinaten. Zur Umrechnung der Angaben wird die inverse Transformation der geklonten Instanz des RenderContext zurückgegeben.

26 com.levigo.jadice.docs.RenderContext

## 9.1. Anlegen/Ändern über das AnnotationCreator Interface

Um eigene Annotationen oder Annotationen mit unterschiedlichen, vordefinierten Eigenschaften interaktiv vom Benutzer anlegen lassen zu können, kann die Schnittstelle AnnotationCreator[27] genutzt werden. Eine vom Integrator erstellte Klasse, die diesem Interface genügt, kann über die statische **setAnnotationCreator**-Methode der Klasse Annotation registriert werden.

Wird nun durch den Benutzer eine Annotation interaktiv mit der Maus angelegt, wird die createAnnotation-Methode der registrierten Instanz der Schnittstelle AnnotationCreator aufgerufen. Der ausgewählte Annotationstyp und die Position der anzulegenden Annotation werden als Parameter übergeben. Die Standard-Annotationstypen stehen als Konstanten der Klasse Annotation statisch zur Verfügung. Näheres dazu ist der API-Beschreibung der Klasse Anntation zu entnehmen.

In der createAnnotation-Methode kann nun eine Annotation angelegt und mit den gewünschten Eigenschaften angepasst werden (siehe Code Beispiel 14). Wird keine Annotation zurückgegeben, wird die dem Annotationstyp entsprechende Standard-Annotation angelegt.

Im folgenden Beispiel wird eine Text-Annotation mit der Grösse 150x50 Pixel und grüner Hintergrundfarbe angelegt.

```
public Annotation createAnnotation(int annotationMode,
  Point atPoint) {
  Annotation anno = null;
  if (annotationMode == Annotation.TEXT) {
    RenderContext rcDef = new RenderContext();
    Dimension size = new
      Dimension(rcDef.deviceToBase(150),
      rcDef.deviceToBase(50));
    TextAnnotation annoText = new
      TextAnnotation(atPoint.x, atPoint.y, size.width,
      size.height, "Neue Anno");
    annoText.setBackgroundColor(Color.green);
    anno = annoText;
  }
  return anno;
}
```

*Code Beispiel 16 – AnnotationCreator*

27 com.levigo.jadice.annotation.AnnotationCreator

# 10. RenderContext

An einigen Stellen dieses Dokuments, insbesondere im folgenden Kapitel, wird Bezug genommen auf die Klasse RenderContext[28].

Zum besseren Verständnis werden an dieser Stelle kurz die Eigenschaften und Aufgaben der Klasse RenderContext vorgestellt.

Im Gegensatz zur Viewer Vs. 2.x werden zur Darstellung in der neuen Viewer Generation keine Images erzeugt, vielmehr rendern sich die Seiten und damit auch alle ihre Segmente selbstständig in gegebene Device-Kontexte, z.B. Monitor, Drucker, etc.

Ein Render-Prozess wird stets begleitet von einer Instanz der Klasse RenderContext. Der RenderContext kapselt verschiedene Darstellungs-Attribute, die bindend den Render Prozess bestimmen.

Die Darstellungs-Attribute unterteilen sich in Direkt-Attribute, wie Zoom, Rotation oder ähnliches, und ProcessSettings[29]. ProcessSettings sind Attribute spezieller Natur, die Darstellungseigenschaften ganz bestimmter Art beschreiben, z.B. AnnotationRenderSettings[30] (Näheres dazu in Kapitel 11).

Des Weiteren bietet der RenderContext Transformationen zur einfachen Umrechnung zwischen Dokument- und Device Koordinatensystem. Näheres dazu: Dokument-Modell aus Kapitel 6, Kapitel 7 und jadice viewer Integrations Dokumentation.

28 com.levigo.jadice.docs.RenderContext
29 com.levigo.jadice.docs.ProcessSettings
30 com.levigo.jadice.annotation.AnnotationRenderSettings

# 11. Sichtbarkeit

Unter bestimmten Umständen kann es sinnvoll sein, die Ansicht aller oder bestimmter Annotationen zu unterdrücken. Ein Beispiel, obwohl die Annotationen im Viewer sichtbar sein sollen, soll der Dokumentendruck ohne Annotationen vollzogen werden.

Der Viewer unterstützt zwei Arten, die Sichtbarkeit von Annotationen zu verändern. Zum einen können alle Annotationen aus-/eingeblendet werden, zum anderen können alle Annotationen eines bestimmten Typs aus-/eingeschaltet werden.

Dazu verwenden Sie die Klasse RenderContext[31], genauer gesagt AnnotationRenderSettings[32], die in der Klasse RenderContext enthalten sind.

```
// RenderContext Instanz vom Viewer
RenderContext rc = viewer.getRenderContext();
AnnotationRenderSettings settings =
  rc.getAnnotationRenderSettings();
System.out.println("Alle Annotationen sichtbar ? "
  + settings.isAnnotationRenderingEnabled());
settings.setAnnotationRenderingEnabled(!settings
  .isAnnotationRenderingEnabled());
System.out.println(" Alle Annotationen sichtbar ? "
  + settings.isAnnotationRenderingEnabled());
```

*Code Beispiel 17 / Annotations-Sichtbarkeit*

## 11.1. Sichtbarkeit aller Annotationen

In Code Beispiel 17 wird zunächst die globale Sichtbarkeit aller Annotationen mittels der Methode **„isAnnotationRenderingEnabled()"** erfragt und auf der Konsole ausgegeben.

Die Sichtbarkeit aller Annotationen kann mit der enstprechenden Methode **„setAnnotationRenderingEnabled(boolean)"** verändert werden.

Hinweis:

Integratoren sollten nach Veränderungen an ProcessSettings den Viewer neu rendern lassen.

BSP: myViewer.repaint();

Zurück zu dem oben genannten Beispiel vom Anfang dieses Kapitels zurück. Soll die Sichtbarkeit von Annotationen im Viewer erhalten bleiben, aber im Dokumentendruck aufgehoben werden, geht man in folgenden Schritten vor:

- Erstellen einer Kopie des Viewer Render Context für den Druckvorgang. Dazu kann der Copy-Konstruktor des Render Context verwendet werden.
- Ändern der Sichtbarkeit von Annotationen in dem kopierten Druck Render Context. Der Viewer Render Context verbleibt damit unverändert und die Annotationen werden im Viewer weiterhin angezeigt.
- Starten der Druckvorgangs, z.B. mittels des Aufrufs

31 com.levigo.jadice.docs.RenderContext
32 com.levigo.jadice.annotation.AnnotationRenderSettings

com.levigo.jadice.docs.printer.PrintManager.executePrintJob(Document, RenderContext) der Klasse PrintManager.

## 11.2. Sichtbarkeit bestimmter Annotationen

Ähnlich wie die Sichtbarkeit aller Annotationen erfragt bzw. gesetzt werden kann, kann auch die Sichtbarkeit von Annotationen bestimmten Typs verändert werden. Der einzige Unterschied besteht darin, eine Bestimmung des Annotationstyps der jeweiligen Methode als Parameter zusätzlich zu übergeben.

Wie im Code Beispiel 18 zu sehen, wird den entsprechenden Methoden zusätzlich das Klassenobjekt des betreffenden Annotationstyps als Parameter übergeben. Damit wird bestimmt, welchen Annotationstyp die Sichtbarkeitsveränderung bzw. Zustandsabfrage betrifft.

```
// RenderContext Instanz vom Viewer
RenderContext rc = viewer.getRenderContext();
AnnotationRenderSettings settings =
  rc.getAnnotationRenderSettings();
System.out.println("Alle Text Annotationen sichtbar ? "
  + settings.isAnnotationRenderingEnabled(
  TextAnnotation.class));
settings.setAnnotationRenderingEnabled(
  TextAnnotation.class, !settings
  .isAnnotationRenderingEnabled(TextAnnotation.class));
System.out.println("Alle Text Annotationen sichtbar ? "
  + settings.isAnnotationRenderingEnabled(
  TextAnnotation.class));
```

*Code Beispiel 18 / Annotations-Sichtbarkeit*

Hinweis:

Integratoren sollten nach Veränderungen an ProcessSettings den Viewer neu rendern lassen.

BSP: myViewer.repaint();

# 12. Eigenschafts-Editoren

Mittels Eigenschafts-Editoren können Attribute einer oder mehrerer Annotationen gleichen Typs verändert werden.

Zur Zeit unterstützt der jadice viewer IBM ContentManager und FileNet kompatible Annotationen. Dementsprechend erlauben die Eigenschafts-Editoren des jadice Pakets Annotations-Veränderungen nur Wertebereichen, die die VI- bzw. die FileNet-Kompabilität nicht verletzen.

In den Demo-Klassen der jadice document platform sind die Eigenschafts-Editoren über das Kontext-Menü zu aktivieren.

Für Integratoren, die Eigenschafts-Editoren in Ihre eigene Applika-tions-Umgebung einbetten möchten, ist die zentrale Klasse AnnotationPropertyEditorFactory[33]. Diese Klasse bietet zur Integration der Editoren zwei Möglichkeiten:

- `getAnnotationPropertyEditorAsPanel(...)` - Diese Methode liefert ein JPanel, in dem die Attribut-Editoren der zu ändernden Annotationen enthalten sind.
- `GetAnnotationPropertyEditorAsDialog(...)` - Diese Methode liefert einen modalen Dialog, in dem die Attribut-Editoren der zu ändernden Annotationen enthalten sind.

## 12.1. Eigene Editoren schreiben

Integratoren ist es freigestellt eigene Eigenschafts-Editoren zu erstellen und einzubinden. Beispielsweise zur Wahrung des Corporate Designs der integrierenden Anwendung oder zur Veränderung der Wertebereiche der Editoren.

Zu jedem Annotations-Attribut gibt es Property-Editoren. Anhand der zu ändernden Annotationen wird die Zusammensetzung der einzelnen Attributs-Editoren zu einem Annotations-Editor ermittelt, der dann über die **AnnotationPropertyEditorFactory** als JPanel oder als Dialog zurückgegeben wird. Bestätigt der Benutzer in dem Editor vollzogene Änderungen, werden diese automatisch in den betroffenen Annotationen übernommen.

Die Basisklasse aller Attributs-Editoren ist AbstractPropertyEditor[34]. AbstractPropertyEditor ist ein JPanel, das einfach in oben genannten Mechanismus integriert werden kann.

Bei der Erstellung eigener AbstractPropertyEditoren bestimmt der Integrator über das Layout und die darin enthaltenen Komponenten. Zu implementieren sind jedoch zwei Methoden:

- **getValue()** - Liefert den aktuellen Wert des betreffenden Attributs. Zur Übertragung des neuen Attribut-Wertes an die Annotation, wenn die Änderungen durch den Benutzer bestätigt werden.
- **setValue(...)** - Setzt den aktuellen Wert der Annotation in dem Editor.

Nach der Erstellung eines eigenen Editors muss dieser noch der PropertyEditorFactory für ein bestimmtes Attribut bekannt gemacht werden.

33 com.levigo.jadice.annotation.edit.AnnotationPropertyEditorFactory
34 com.levigo.jadice.annotation.edit.AbstractPropertyEditor

Dazu ist kein weiterer Programmieraufwand nötig.

Im Package com.levigo.jadice.resources.properties sind alle die jadice document platform beschreibenden Konfigurations-Dateien abgelegt. Die Datei „AnnotationEdit.properties" beinhaltet alle Angaben zu Annotations Eigenschafts-Editoren.

Innerhalb dieser Datei befinden sich unter dem Abschnitt „Annotation Properties" verschiedene Key-Value Pairs. Attribut-Editoren werden angemeldet unter dem Schlüssel „attributName.editor". Der anzugebende Wert ist der Klassenname des neu erstellten Editors. Attribut-Editoren werden über Reflektion instantiiert. Unbedingt zu beachten ist die korrekte Angabe des Klassennamens. Näheres zu der Datei AnnotationEdit.properties befindet sich in Kapitel 12.

```
rotation.editor=mein.package.MeinRotationsEditor
```

*Code Beispiel 19 / Eigener Editor*

Hinweis:

Es ist angeraten, diese Konfigurations-Dateien zu kopieren und im Klassen-Pfad vor den jadice viewer zu legen. Damit werden gemachte Änderungen erkannt, aber die ursprüngliche Konfiguration bleibt unangetastet.

Hinweis:

Bitte beachten Sie, dass die Konfigurations-Dateien in internationalisierten Varianten vorliegen. Änderungen der Konfiguration sollten stets in allen Varianten getätigt werden.

# 13. Die Konfigurations-Datei: AnnotationEdit.properties

Hinweis:

Das Suffix der Einstellungen für VisualInfo / ImagePlus Annotationen beginnt mit „**vi.**", das für FileNet Annotationen mit „**fn.**".

Wenn kein Suffix definiert ist, dann wird die VisualInfo / ImagePlus Einstellung benutzt.

Mittels der Konfigurations-Datei AnnotationEdit.properties sind Einstellungen für folgende Bereiche möglich:

- Welche Eigenschaften sind für welchen Annotationstyp veränderbar?
- Für welche Attribute soll welcher Editor benutzt werden?
- Text-Ressourcen für GUI Elemente der Attributs-Editoren/Dialog

Zur besseren Orientierung sind im Folgenden die Einstellungen näher erläutert. Alle Setzungen werden bestimmt über Key-Value Pairs.

## 13.1. Welche Eigenschaften sind für welchen Annotationstyp veränderbar?

Diese Angaben befinden sich unter dem Titel „Annotation Editor Mapping".

Pro Annotationstyp kann bestimmt werden, welche Attribute dieser Art von Annotation veränderlich sein sollen. Dabei entspricht der Key der qualifiziert benannten Annotations-Klasse, das Value ist eine Komma getrennte Liste der veränderlichen Attribute.

```
#
# Annotation Editor Mapping
#
...
com.levigo.jadice.annotation.FreehandAnnotation=foregroundColor,lineWidth
```

*Code Beispiel 20 / Annotation Editor Mapping*

Diese Zeile besagt, dass die veränderlichen Attribute der Freehand-Annotation die Vordergrundfarbe und die Linien-Breite sind.

## 13.2. Für welche Attribute soll welcher Editor benutzt werden?

Diese Angaben befinden sich unter dem Titel „Editors".

An dieser Stelle können Sie bestimmen, welcher Editor zur Veränderung eines Attributs verwendet werden soll. Der Key setzt sich zusammen aus dem Attributs-Namen und dem Suffix „**.editor**". Das Value entspricht der qualifiziert benannten Editor-Klasse. Da Editoren mittels Reflektion erstellt werden, achten Sie bitte an dieser Stelle auf korrekte Schreibweise des Klassennames.

```
#
# Editors
#
...
foregroundColor.editor=com.levigo.jadice.annotation.edit.BaseColorEditor
```

*Code Beispiel 21 / Editors*

Durch diese Angabe wird die Klasse BaseColorEditor als Editor für die Vordergrundfarbe bestimmt.

## 13.3. Text-Ressourcen für GUI Elemente

Diese Angaben befinden sich unter dem Titel „Titles and Strings".

In diesem Abschnitt definieren sich Eigenschafts-Editoren Text-Ressourcen für die von ihnen benutzten GUI-Elemente. Es handelt sich dabei bspw. um die Feldbenennung der Attribut-Eingabe-Felder, einen Dialog-Titel, aber auch um Fehlermeldungstexte.

```
#
# Titles and Strings
#
...
foregroundColor.title=Vordergrund Farbe
```

*Code Beispiel 22 / Titles and Strings*

Hinweis:

Es ist angeraten, diese Konfigurations-Dateien zu kopieren und im Klassen-Pfad vor den jadice viewer zu legen. Damit werden gemachte Änderungen erkannt, aber die ursprüngliche Konfiguration bleibt unangetastet.

Hinweis:

Bitte beachten Sie, dass die Konfigurations-Dateien in internationalisierten Varianten vorliegen. Änderungen der Konfiguration sollten stets in allen Varianten getätigt werden.

Hinweis:

Veränderungen an dieser Konfigurations-Datei könnten, je nach angebundenem Archivsystem und damit verbundener Annotationsart, zu Kompatibilitäts-Problemen führen.

# 14. Die Konfigurations-Datei: AnnotationInit.properties

Hinweis:

Das Suffix der Einstellungen für VisualInfo / ImagePlus Annotationen beginnt mit „**vi.**", das für FileNet Annotationen mit „**fn.**".

Wenn kein Suffix definiert ist, dann wird die VisualInfo / ImagePlus Einstellung benutzt.

Mittels der Konfigurations-Datei AnnotationInit.properties sind Einstellungen möglich, um das Aussehen einer neu angelegten Annotation zu definieren:

- Farbdefinition für Vorder-/Hintergrund und Rand (alle Annotationstypen)
- Einstellung von Eigenschaften:
  - Transparenz
  - Hintergrund füllen
  - Randdarstellung
  - Linienstärke
  - Rotation
  - Darstellung der Annotation als Icon
  - Darstellung der Annotationsnummern
- Mindestgrösse einer neu angelegten Annotation (alle Annotationstypen)
- Schriftgrösse und Stil (Note-, Stamp- und Text-Annotation)
- Mindestgrösse des Texteingabefensters (Note-, Stamp- und Text-Annotation)

## 14.1. Allgemeine Eigenschaften (VisualInfo / ImagePlus und FileNet Annotationen)

Allgemeine Eigenschaften sind unabhängig vom Annotationstyp und gelten für VisualInfo / ImagePlus und FileNet Annotationen. Es wird kein Suffix bei den Parameternamen verwendet.

**Annotationsnummern:**

**annotation.show-number-on-start=**<Wert>

Parameter, ob die Annotationsnummern beim Start sofort angezeigt werden sollen.

Wert: true = Nummern werden beim Start angezeigt.
false = Nummern werden nicht angezeigt.

**annotation.number-display-mode=<Wert>**

Parameter zur Definition der Nummerndarstellung (Verhalten beim Zoomen).

Wert: 0 = Nummerngrösse ist abhängig vom Zoomfaktor.
1 = Nummerngrösse wird nicht vergrössert, wenn der Zoomfaktor grösser 100 ist.

2 = Nummerngrösse wird nicht verkleinert, wenn der Zoomfaktor kleiner 100 ist.
3 = Nummerngrösse ist Zoomfaktor unabhängig.

**annotation.number-display-size=<Wert>**

Parameter zum Einstellen der Nummerngrösse.

Wert: Grösse der Schrift.

## 14.2. Annotationseigenschaften

Die Definition setzt sich wie folgt zusammen:

<**suffix**>.<**annotationstyp**>.<**parameter**>

**suffix** = fn oder vi

**annotationstyp** = Annotationstyp (Paketname + Klassenname)

Beschreibung der möglichen Parameter:

- **defaultSize** (Typ = Dimension, Standard=1,1)
  Grösse beim Anlegen einer Annotation.
  -> alle Annotationen ausser Stamp

- **editFrameMinSize** (Typ = Dimension, Standard=1,1)
  Grösse des Texteingabefensters.
  -> Note, Text, Stamp

- **foregroundColor** (Typ = Color, Standard=Color.yellow)
  Vordergrundfarbe.
  -> Arrow, Ellipse, Freehand, Highlight, Line, Rectangle, Stamp, Text

- **backgroundColor** (Typ = Color, Standard=Color.white)
  Hintergrundfarbe.
  -> Arrow, Ellipse, Freehand, Highlight, Line, Rectangle, Stamp, Text

- **transparent** (Typ = boolean, Standard=false)
  Hintergrundtransparenz.
  -> Rectangle, Stamp, Text

- **iconify** (Typ = boolean, Standard=false)
  Icondarstellung.
  -> Note

- **linePainted** (Typ = boolean, Standard=true)
  Randlinie zeichnen.
  -> Arrow, Ellipse, Freehand, Highlight, Line, Rectangle, Stamp, Text

- **borderColor** (Typ = Color, Standard=Color.yellow)
  Randfarbe.
  -> Ellipse, Rectangle, Stamp, Text

- **lineWidth** (Typ = int , Standard=3)
  Linienstärke.
  -> Arrow, Ellipse, Freehand, Line, Rectangle, Stamp, Text

- **filled** (Typ = boolean, Standard=false)
  Hintergrund füllen.
  -> Ellipse, Rectangle, Stamp, Text

- **fontFace** (Typ = String, Standard=Sansserif)
  Schriftart.
  -> Note, Stamp, Text

- **fontSize** (Typ = int, Standard=36)
  Schriftgrösse.
  -> Note, Stamp, Text

- **fontBold** (Typ = boolean, Standard=false)
  Bold Schriftstil.
  -> Note, Stamp, Text

- **fontItalic** (Typ = boolean, Standard=false)
  Italic Schriftstil.
  -> Note, Stamp, Text

- **headAngle** (Typ =int, Standard=20)
  Winkel der Pfeilspitze.
  -> Arrow

- **headLength** (Typ =int, Standard=25)
  Länge der Pfeilspitze.
  -> Arrow

- **allowResize** (Typ =boolean, Standard=false)
  Grösse ändern.
  -> Freehand

- **gap** (Typ = int, Standard=10)
  Abstand Schrift zum Rand.
  -> Stamp

- **rotation** (Typ = int, Standard=20)
  Rotation.
  -> Stamp

- **userName** (Typ = String, Standard=unknown)
  Benutzername.
  -> History

- **pattern** (Typ = String, Standard=<---[Created by <user> at <date> <time>]--->)
  Muster für die Infozeile.
  <user> = Wird durch Benutzername ersetzt.
  <date> = Wird durch das aktuelle Datum ersetzt.
  <time> = Wird durch die aktuelle Zeit ersetzt.
  -> History

- **date** (Typ = String, Standard=DDMMYYYY)
  Darstellung des Datums in der Infozeile.
  Mögliche Muster: DDMMYYYY, YYYYMMDD
  -> History

- **time** (Typ = String, Standard=24H_DEFAULT)
  Darstellung der Zeit in der Infozeile.
  Mögliche Muster: 12H_AM_PM, 24H_DEFAULT
  -> History

- **patternPosition** (Typ = String, Standard = PATTERN_HISTORY_AT_BOTTOM)

  Position der Infozeile.

  Mögliche Muster: PATTERN_HISTORY_AT_BOTTOM , PATTERN_HISTORY_AT_TOP

  -> History

**Beispiel anhand einer VisualInfo / ImagePlus TextAnnotation:**

```
vi.com.levigo.jadice.annotation.TextAnnotation.lineWidth=1
vi.com.levigo.jadice.annotation.TextAnnotation.foregroundC
olor=255
vi.com.levigo.jadice.annotation.TextAnnotation.defaultSize
=50,50
vi.com.levigo.jadice.annotation.TextAnnotation.text=Text-
Annotation
vi.com.levigo.jadice.annotation.TextAnnotation.editFrameMi
nSize=100,75
```

*Code Beispiel 23 / Annotationseigenschaften*

## 14.3. Annotationstexteinstellungen

Die Textdarstellung der textbasierenden Annotationen kann angepasst werden, die folgenden Einstellungen zur Textanpassung gelten für Note-, Text- und History-Annotationen.

Bei der Textausrichtung werden nur ganze Worte berücksichtigt.

Anmerkung zu den Beispielen: <CR> steht für einen eingegebenen Zeilenumbruch im Eingabefenster, die # stellen den Fensterrand dar.

- **adjustTextToWindow=false**

  Text wird NICHT an die Anzeigefenstergrösse angepasst.

Mit dem keepEditText Parameter kann das Verhalten gesteuert werden:

- **keepEditText=false (Default)**

  Der Text wird nicht angepasst, der Text aus dem Texteingabefenster wird an den Zeilenumbrüchen ausgerichtet.

```
Texteingabefenster:
######################
#Dies ist eine Zeile #
#mit Text.           #
#                    #
######################

Anzeigefenster:
######################
#Dies ist eine Zeile #
#                    #
#                    #
```

```
######################
```

*Annotations-Darstellung 1 / Ohne Umbruch*

```
Texteingabefenster:
######################
#Dies ist die 1.     #
#Zeile mit Text.<CR> #
#Hier kommt Zeile 2. #
######################

Anzeigefenster:
######################
#Dies ist die 1. Zeil#
#Hier kommt Zeile 2. #
#                    #
######################
```

*Annotations-Darstellung 2 / Mit Umbruch*

- **keepEditText=true**

  Der Text wird aus dem Texteingabefenster übernommen, bei einem Umbruch im Eingabefenster wird ein Zeilenumbruch generiert.

```
Texteingabefenster:
######################
#Dies ist eine Zeile #
#mit Text.           #
#                    #
######################

Anzeigefenster:
######################
#Dies ist eine Zeile #
#mit Text.           #
#                    #
######################
```

*Annotations-Darstellung 3 / Ohne Umbruch*

```
Texteingabefenster:
######################
#Dies ist die 1.     #
#Zeile mit Text.<CR> #
#Hier kommt Zeile 2. #
######################

Anzeigefenster:
######################
```

```
#Dies ist die 1.     #
#Zeile mit Text.     #
#Hier kommt Zeile 2. #
######################
```

*Annotations-Darstellung 4 / Mit Umbruch*

- **adjustTextToWindow=true (Default)**

  Der keepEditText Parameter sollte hier nicht verwendet werden (keepEditText=false), da sonst evtl. noch zusätzliche Zeilenumbrüche erzeugt werden, die das Verhalten undurchsichtig erscheinen lassen können.

  Der Text wird beim Vergrössern/Verkleinern der Annotation an die Grösse angepasst. Der im Eingabefenster eingegebene Text wird nicht verändert, d.h. der Text wird NUR bei der Anzeige verändert.

  Mit dem **adjustTextToWindowNewLineMode** Paramter kann zusätzlich das Verhalten von eingegebenen Zeilenumbrüchen gesteuert werden (funktioniert nur in Verbindung mit adjustTextToWindow=true).

- **adjustTextToWindowNewLineMode=0 (Default)**

  Eingegebene Zeilenumbrüche werden bei der Anzeige NICHT berücksichtigt.

```
Texteingabefenster:
######################
#Dies ist eine Zeile #
#mit Text.           #
#                    #
######################

Anzeigefenster vergrössert:
################################
#Dies ist eine Zeile mit Text.  #
#                              #
#                              #
################################

Anzeigefenster verkleinert:
##################
#Dies ist eine   #
#Zeile mit Text. #
#                #
##################
```

*Annotations-Darstellung 5 / Ohne Umbruch*

```
Texteingabefenster:
######################
#Dies ist die 1.     #
#Zeile mit Text.<CR> #
#Hier kommt Zeile 2. #
```

```
######################

Anzeigefenster vergrössert:
#######################################
#Dies ist die 1. Zeile mit Text. Hier #
#kommt Zeile 2.                       #
#                                     #
#######################################

Anzeigefenster verkleinert:
##################
#Dies ist die 1. #
#Zeile mit Text. #
#Hier kommt Zeile#
##################
```

*Annotations-Darstellung 6 / Mit Umbruch*

- adjustTextToWindowNewLineMode=1

Eingegebene Zeilenumbrüche werden bei der Anzeige berücksichtigt.

```
Texteingabefenster:
######################
#Dies ist eine Zeile #
#mit Text.           #
#                    #
######################

Anzeigefenster vergrössert:
#################################
#Dies ist eine Zeile mit Text.  #
#                               #
#                               #
#################################

Anzeigefenster verkleinert:
##################
#Dies ist eine   #
#Zeile mit Text. #
#                #
##################
```

*Annotations-Darstellung 7 / Ohne Umbruch*

```
Texteingabefenster:
######################
#Dies ist die 1.     #
#Zeile mit Text.<CR> #
#Hier kommt Zeile 2. #
######################
```

```
Anzeigefenster vergrössert:
########################################
#Dies ist die 1. Zeile mit Text.      #
#Hier kommt Zeile 2.                  #
#                                     #
########################################

Anzeigefenster verkleinert:
###########
#Dies ist #
#die 1.   #
#Zeile mit#
#Text.    #
#Hier     #
#kommt    #
#Zeile 2. #
#         #
###########
```

*Annotations-Darstellung 8 / Mit Umbruch*

# 15. Dokumenthistorie

| Version | Datum | Autor | Änderungen |
|---|---|---|---|
| 1.0 | 02.09.03 | Carolin Köhler | Kapitel 1 - 5 |
| | 03.09.03 | Carolin Köhler | Kapitel 6 |
| | 04.09.03 | Carolin Köhler | Kapitel 7, Kapitel 8 teilweise |
| | 05.09.03 | Carolin Köhler | Kapitel 8 - 11 |
| | 08.09.03 | Carolin Köhler | Kapitel 12 |
| 1.1 | 08.12.03 | M. Grossmann | Erweiterungen zu FileNet Annotationen<br>Kapitel 3, 4, 5, 6, 12, 13 |
| 1.2 | 20.01.04 | M. Grossmann | Änderungen zu FileNet Annotationen<br>Kapitel 4 |
| 1.3 | 22.04.04 | M.Grossmann | Änderungen in Kapitel 3, 5, 6<br>Erweiterungen in Kapitel 4, 14 |
| 1.4 | 23.01.06 | M. Grossmann | Änderungen in Kapitel 5<br>Erweiterungen in Kapitel 9 |
| 2.0 | 06.02.06 | Carolin Köhler<br>M. Grossmann | Neu überarbeitet |
| 4.0 | 30.11.07 | M. Grossmann | Erweiterung FileNet P8 Annotationen |
| 4.1 | 29.02.08 | M. Grossmann | Neu überarbeitet |
| 4.2 | 18.02.09 | | |