World of
Documents

Benutzerdokumentation

World Of Documents: Benutzerdokumentation

Jeremias Märki

Stand: 23.11.2016

Inhaltsverzeichnis

Abbildungsverzeichnis

Liste der Beispiele

Einleitung

World Of Documents ist eine Serverlösung für das Output Management. Es umfasst Funktionen zur Dokumentenerzeugung und -verarbeitung.

Kapitel 1. Installation

Dieses Kapitel beschreibt die Installation von World Of Documents (kurz: WOD).

1.1. Systemvoraussetzungen

Nachfolgend sind die Mindestvoraussetzungen für den Betrieb von World Of Documents aufgelistet.

- Eine Java VM (JVM, JavaSE 1.6 oder höher). Eine Oracle JVM ist gegenüber OpenJDK zu bevorzugen, hauptsächlich wegen kleineren Problemen im Bereich Grafik (Java2D). Der Einsatz einer Server JVM ist empfohlen (Standard auf 64bit Maschinen).
- Empfohlen wird die Zuweisung von mindestens 256 MB RAM an die JVM. Mit minimaler Ausstattung ist ein Betrieb mit ca. 100MB möglich.
- Die Installation benötigt knapp 100MB Disk-Speicher. Für den Einsatz einer Message Queue oder eines Content Repository muss entsprechend weiterer Speicherplatz zur Verfügung stehen.

Je nach Einsatzzweck und Ausbaustufe werden natürlich zusätzliche Ressourcen benötigt. Formatierungs- und Konversion-Operationen können sehr rechenintensiv sein.

WOD wurde primär auf Windows und Linux entwickelt und getestet, sollte aber unverändert auch auf BSD, Solaris und anderen Systemen mit JavaSE 1.6 Unterstützung lauffähig sein.

1.2. Installationstypen

WOD kann auf verschiedene Arten installiert werden. Da es sich bei WOD um ein OSGi-basiertes System handelt, sind verschiedenste Deployment-Formen denkbar. Im Folgenden werden die unterschiedlichen Typen aufgezeigt.

1.2.1. Skeleton-Container mit Initial Provisioning

In dieser Deployment Form wird auf der Zielmaschine nur eine Skeleton-Applikation installiert, welche die eigentliche Applikation über das Netzwerk von einem Deployment-Server herunterlädt. Diese Form lässt es zu, später einfach, neue Updates nachzureichen. Der Umfang der installierten Komponenten wird dabei vom Deployment-Server bestimmt und kann von der Lizenz abhängen. Der Management Teil der Applikation versucht bei jedem Neustart oder durch explizites Auslösen Kontakt mit dem Deployment-Server aufzunehmen um sich zu aktualisieren.

Der Vorteil hierbei ist der minimale Installationsaufwand auf Server-Seite und die automatische Aktualisierung. Konfigurationen können ebenfalls über den Deployment-Server verteilt werden. Mehrere Installationen in einer Umgebung (oder für mehrere Kunden) sind hierdurch ebenfalls einfach möglich und ermöglichen eine simultane Aktualisierung aller Installationen.

Der Nachteil ist, dass u.U. eine Internet Verbindung verfügbar sein muss.

1.2.1.1. Technische Details

WOD läuft in einer OSGi-Umgebung. Der Skeleton-Container enthält praktisch nur einen leichtgewichtigen Container für die Initial-Konfiguration und für das Hochfahren des OSGi Frameworks. Als Bundles werden lediglich der ConfigAdmin und der Initial Provisioning Service installiert. Initial Provisioning holt sich ab einem konfigurierten Ort den Management Agent, welcher schlussendlich die Applikations-Bundles herunterlädt, installiert und gegebenenfalls aktualisiert.

1.2.2. Einfache Installation

In dieser Deployment Form wird WOD mit sämtlichen Applikations-Bundles zusammen installiert. Hierbei wird keine Management Agent verwendet. Aktualisierungen der Applikationen geschehen ebenfalls weitgehend manuell. Diese Installationsform kann gewählt werden, wenn das Deployment über einen separaten Server nicht möglich oder nicht erwünscht ist.

1.2.2.1. Technische Details

Sämtliche Bundles sind in diesem Fall unter dem Verzeichnis `bundles/<start-level>/` abgelegt, wobei <start-level> eine Ganzzahl ist und den OSGi Start Level der darin enthaltenen Bundles angibt.

1.2.3. Embedding

Grundsätzlich kann WOD als Set von Bundles in einer beliebigen OSGi-Umgebung installiert werden. Dies bedarf jedoch einiger Vorsicht, da es durch schlecht geschriebene OSGi Bundles zu Beeinträchtigungen beim Betrieb geben kann. Diese Installationsform wird offiziell nicht unterstützt, ist aber aufgrund des OSGi Ansatzes grundsätzlich möglich.

Umgekehrt können natürlich auch WOD-fremde OSGi-Komponenten wie z.B. Apache Camel oder Apache ActiveMQ in die WOD-Umgebung hinein installiert werden, um z.B. den Funktionsumfang zu erweitern. Auch hier muss das Zusammenspiel der Komponenten sorgfältig getestet werden.

1.2.4. Weitere Installationstypen

Denkbar sind weitere Deployment Formen z.B. mittels Deployment Packages (aus dem OSGi Compendium) oder via Apache ACE.

1.3. Verzeichnis-Layout

Das standardmässige Verzeichnis-Layout von WOD sieht folgendermassen aus:

- `<WOD-HOME>/bin`: Enthält Scripts und plattform-spezifische Binaries zum Start von WOD.
- `<WOD-HOME>/etc`: Enthält die Basis-Konfiguration für WOD.
- `<WOD-HOME>/lib`: Enthält das minimale Set an Java Bibliotheken für den Start des WOD Containers.
- `<WOD-HOME>/lib/endorsed`: Enthält Java Bibliotheken, welche die Standard-Komponenten aus der Java Runtime überschreiben. Beispielsweise enthalten die XML Parser und XSLT Prozessoren von verschiednen Java Runtimes verschiedene Bugs, weshalb WOD bestimmte Versionen von Apache Xerces-J und Apache Xalan-J verwendet. Dieses Verzeichnis wird per Kommandozeilen-Parameter gesetzt, z.B. **-Djava.endorsed.dirs="%JAVA_HOME%\jre\lib\endorsed;%JAVA_HOME%\lib\endorsed;%WOD_HOME%\lib\endorsed"**.
- `<WOD-HOME>/lib/ext`: Enthält das minimale Set an Java Bibliotheken für den Start des WOD Containers. Dieses Verzeichnis wird ebenfalls als Kommandozeilen-Parameter gesetzt, z.B. **-Djava.ext.dirs="%JAVA_HOME%\jre\lib\ext;%JAVA_HOME%\lib\ext;%WOD_HOME%\lib\ext"**.
- `<WOD-HOME>/bundles`: Enthält verschiedene OSGi Bundles, die beim Start von WOD aktiviert werden. Dort befindt sich entweder die gesamte WOD Funktionalität oder nur ein sogenannter Management Agent, der das Nachladen der WOD Funktionalität von einem Deployment Server übernimmt.
- `<WOD-HOME>/data` (optional): Falls die Daten zusammen mit der Applikation gehalten werden sollen, befindet sich das Daten-Verzeichnis hier. Die Daten und Logs können aber auch an einem anderen Ort gehalten werden. Dazu muss lediglich die Konfiguration angepasst werden.

1.4. Beispiel: "Einfache" Installation, Schritt für Schritt

Dieser Abschnitt zeigt eine Installation Schritt für Schritt am Beispiel der "einfachen" Installation (siehe Abschnitt 1.2.2, „Einfache Installation“) auf einem Windows Server. Für diese Form der Installation wird World Of Documents als ZIP oder TAR Datei zur Verfügung gestellt. Hier der übliche Ablauf:

1. Die ZIP Datei mit der Installation auf den Server hochladen und dort in ein frisches Verzeichnis entpacken. Dies ergibt die oben beschriebene Verzeichnisstruktur.

2. Auf Wunsch neueste Server JVM von Oracle[1] herunterladen. Die Server JVM kann man beispielsweise in ein Unterverzeichnis der WOD Installation entpacken. Allerdings ist das nicht zwingend, solange auf der Maschine ein JavaSE 6.0 oder neuer installiert ist.
3. Jetzt muss ein Datenverzeichnis angelegt werden, wo WOD z.B. Datenbanken oder Log-Dateien speichern kann. Dieses ist standardmässig auf das `data` Verzeichnis unter dem Installationsverzeichnis von WOD eingestellt. Soll das Datenverzeichnis woanders zu liegen kommen, muss die Konfiguration angepasst werden, und zwar muss entweder die `WOD_DATA` Environment Variable gesetzt werden (im Betriebssystem oder über das wod-variables.bat beim Windows Service), oder es wird das "app.data" System Property gesetzt.
4. Zur Steuerung, welche JVM zur Ausführung von WOD verwendet werden soll, kann die `JAVA_HOME` Environment Variable gesetzt werden. Dies kann sowohl direkt im Betriebssystem oder in `bin/wod-variables.bat` (Unix: `bin/wod-variables`) gesetzt werden. Dieser Schritt ist in vielen Fällen nicht notwendig.
5. Starten Sie als Administrator (wichtig!) eine Kommandozeile. Gehen Sie in WOD's `bin` Verzeichnis und führen Sie dort **wod install** aus, um den Windows Service für WOD zu installieren (siehe auch Abschnitt 1.5, „Installation als Windows Service“).
6. Nun ist der Windows Service bereit zum Start. Es ist zu beachten, dass der Windows Service bei Installation auf "manuell" eingestellt ist, also bei Neustart nicht automatisch startet. Wenn WOD also sauber läuft, bitte auch "automatisch" einstellen.

   Müssen direkt Windows Drucker angesprochen werden, muss WOD statt als "Local System" unter einem technischen User Account laufen, der die notwendigen Rechte hat. Siehe auch Abschnitt 1.5.2.1, „User für den WOD Service“.
7. Nach dem Start von WOD kann über den Browser geprüft werden, ob es ordnungsgemäss funktioniert. Standardmässig kann man die WOD System Console über `http://localhost:40080/system/console/` aufrufen (ohne spezielle Konfiguration: Username: admin, Passwort: admin). Beispielsweise müssen unter "Bundles" sämtliche Bundles auf "Active" oder "Fragment" stehen. Ist dem nicht so, hat WOD nicht sauber gestartet. Alternativ oder zusätzlich kann das Applikations-Log im Daten-Verzeichnis eingesehen werden, welches insbesondere bei Problemen nützliche Informationen bereitstellt.
8. Eine zusätzliche Möglichkeit zu kontrollieren, ob WOD sauber läuft, ist über die Administrations-Applikation, normalerweise unter `http://localhost:40080/wod-admin/` zu finden. Dort kann beispielsweise die Liste der aktiven Dokumententypen geprüft werden (wenn diese ordnungsgemäss konfiguriert sind).

1.5. Installation als Windows Service

Java Applikationen können auf verschiedene Arten als Windows Services betrieben werden. Beispielhaft soll hier der Ansatz mit **procrun** von Apache Commons Daemon[2] vorgestellt werden.

WOD wird standardmässig über einen an Apache Karaf[3] angelehnten, aber vereinfachten Container gestartet, der u.a. das `Daemon` Interface von Apache Commons Daemon implementiert. Für **procrun** existiert je eine Start und Stop Klasse.

Für die Installation eines Service werden Administrator-Rechte benötigt. Das Batch Script `wod.bat` erlaubt eine einfach Installation. Vorher muss die JAVA_HOME Umgebungsvariable gesetzt sein. Mit **wod install** ("install" in Kleinbuchstaben!) lässt sich der Service installieren. Danach können mit **wod manage** alle Einstellungen kontrolliert und allenfalls angepasst werden (z.B. die Auswahl der JVM oder die Speichereinstellungen). Auch für dieses Programm werden Administrator-Rechte benötigt.

Die Auswahl der JVM kann u.U. wichtig sein. Im "Java" tab kann "Use Default" ausgeschalten werden und unter "Java Virtual Machine" eine bestimmte JVM ausgewählt werden. Die notwendige `jvm.dll` ist üblicherweise in der Java Installation unter `bin/server/jvm.dll` (oder `jre/bin/server/jvm.dll`) gefunden werden.

[1] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[2] http://commons.apache.org/daemon
[3] http://karaf.apache.org

Nach der Installation können allfällige Ausgaben auf die Konsole in `%APP_DATA%/log/commons-daemon.log` eingesehen werden.

Die De-Installation passiert über **wod uninstall** oder durch manuelle Entfernung in der Windows Registry.

1.5.1. Drucker-Installation

Bei der Installation von WOD als Windows Service ist zu beachten, dass sämtliche Drucker, die über das Java Printing System (JPS) angesprochen werden, auf dem Server installiert werden, und zwar so, dass der (technische) Benutzer, unter dem WOD läuft, die Berechtigung hat, auf diese Drucker zuzugreifen. Das ist üblicherweise auf Windows Server mit dem Standard-User "LocalSystem" nicht der Fall. In einem solchen Fall empfiehlt es sich einen speziellen "technischen" User anzulegen, der auf die installierten Drucker zugreifen kann.

Die verfügbaren Windows Drucker werden im WOD Admin Client unter dem Menupunkt "Java Printing System" angezeigt. Werden Drucker dort nicht angezeigt, handelt es sich in den meisten Fällen um ein Berechtigungsproblem.

Werden neue Druckertreiber installiert oder die Berechtigungen für den Drucker geändert, muss WOD neu gestartet werden. Dies ist eine Limitierung des Java Printing System. Es wurde ursprünglich für die Benützung durch Desktop-Programme konzipiert und ist nicht für den Server-Betrieb ausgelegt. Deshalb ist aufgrund der fehlenden Flexibilität, wo immer möglich, die Benützung von LPR oder JetDirect zur Ansteuerung von Druckern empfohlen.

1.5.2. Problembehandlung beim Windows Service

1.5.2.1. User für den WOD Service

Standardmässig läuft WOD mit dem User "Local System". Es sollte auch als "Network Service" laufen, welcher weniger Freiheiten geniesst. Oder man kann auch einen speziellen technischen User Account für WOD einrichten (was z.B. unter Unix üblich ist).

Ein technischer User ist insbesondere notwendig, wenn auf einem Windows Server direkt Windows-Drucker angesteuert werden sollen. "Local System" erhält nicht automatisch Berechtigung auf diese Drucker.

1.5.2.2. Probleme beim Start von WOD auf Windows Server 2008

Folgendes gilt insbesondere, wenn die Embedded Datenbank Apache Derby beim Start von WOD auf Windows Server 2008 hängen bleibt (change lock blockiert, Datenbank-Dienste funktionieren nicht). Bei Problemen z.B. mit der eingebetteten Apache Derby Datenbank ist es am Einfachsten das `data` Verzeichnis zu leeren (nicht löschen, da WOD das `data` Verzeichnis nicht automatisch anlegt!). Alternativ kann man auch nur das `derby` Verzeichnis löschen, wenn das Problem mit dem Löschen des `data` Verzeichnisses nicht gelöst werden kann. Es ist noch nicht klar, was genau diese Probleme verursacht. Es könnte mit Datei-Permissions zu tun haben oder ev. auch eine Art von Race-Condition. Vorsicht: mit diesen Work-Arounds können u.U. wertvolle Daten verloren gehen! Dieser Ansatz empfielt sich nur bei einer Erst-Installation.

Auch schon vorgekommen ist, dass Apache Commons Daemon die Java VM gar nicht starten konnte. Dies trat auf einer 64bit Machine (Windows Server 2008) auf. Das Problem konnte dadurch behoben werden, dass der Service erst deinstalliert wurde. Anschliessend wurde eine 32bit JVM (6.0_xx) installiert und for dem **wod install** auf der Konsole das Kommando **SET WOD_EXE=wod.exe** ausgeführt wurde. Das überschreibt die automatische Erkennung der Architektur (32bit vs. 64bit) und ruft damit die 32bit Version von Apache Commons Daemon auf. Es kann also sein, dass auf gewissen Maschinen Probleme mit der standardmässigen JVM (Java Runtime Environment) bestehen, die Apache Commons Daemon beeinflussen. Es kann deshalb nützlich sein, ein weiteres Java Runtime Environment (JRE) an

anderer Stelle zusätzlich zu installieren und in der Service Konfiguration darauf zu verweisen. Im obigen Beispiel war eine 32bit JVM zur Hand, aber die Benützung einer separaten 64bit JVM hätte unter Umständen gereicht.

1.5.2.3. Probleme mit Spaces in Dateipfaden

Obwohl `wod.bat` entsprechend vorbereitet ist, können u.U. Probleme mit Spaces in Dateipfaden (z.B. wegen `C:\Program Files\` auftreten. Deshalb sollten bei Problemen die Einstellungen des Service über **wod manage** überprüft werden. Entweder passt man die Werte automatisch an oder deinstalliert den Service, ändert `wod.bat` und installiert den Service neu.

1.5.2.4. Probleme mit UNC Pfaden

Sendet man Jobs an WOD, werden idealerweise URIs und URLs benützt um externe Ressourcen (Dateien) zu referenzieren. Als Vereinfachung werden auch einfache Dateinamen angenommen, welche intern in sogenannte "file" URLs umgewandelt werden. Java hat zum Teil Mühe mit UNC Pfaden, weshalb von deren Benützung grundsätzlich abgeraten wird. Sollen sie aber dennoch verwendet werden, gilt es zu beachten, dass die SMB Share Namen keine Underscores ("_") enthalten. Diesbezüglich gibt es sowohl in Java 6 wie auch 7 einen Bug. Shares wie z.B. "\\SERVER01\app_data" werden also Probleme verursachen.

Ist die Änderung des SMB Share Namen nicht möglich, kann auf einen Trick zurückgegriffen werden. Zusätzlich zum eigentlichen Share, kann ein zusätzlicher eingerichtet werden, der keinen Underscore enthält. Falls gewünscht, kann er über ein angehängtes Dollar-Zeichen ("$") versteckt werden. Im obigen Beispiel würde man also zusätzlich einen Share namens "\\SERVER01\appdata$" erstellen. Anschliessend kann im "etc" Verzeichnis von WOD (oder in einem anderen Deployment-Verzeichnis) der XML Katalog (`catalog.xml`) um eine Zeile erweitert werden. Ein Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">

  <!-- [..] -->

  <rewriteURI uriStartString="file://SERVER01/app_data/"
              rewritePrefix="file://SERVER01/appdata$/"/>

</catalog>
```

Hiermit wird intern die generierte "file" URL umgeschrieben und der Share Name ersetzt. Wann immer also von aussen der Share "app_data" angesprochen wird, verwendet WOD intern stattdessen "appdata$".

Gibt es beim Zugriff auf "file" URLs (also z.B. bei der Verwendung von UNC Pfaden) Zugriffsverweigerungen ("Access denied") sind die Berechtigungen auf dem Share zu überprüfen. Bei Bedarf sind dem User, unter dem WOD läuft, die nötigen Berechtigungen zuzuweisen. Wenn WOD z.B. als "LocalSystem" läuft, ist in den Share-Berechtigung der Name "Local" einzutragen.

1.6. Allgemeine Fehlerbehandlung

1.6.1. Nicht alle Bundles sind aktiv

In seltenen Fällen kann es vorkommen, dass nicht alle Bundles sauber gestartet werden. Dies kann zum Beispiel gesehen werden, wenn in der WOD System Console unter "Bundles" nicht alle Bundles auf "Active" oder "Fragment" stehen.

Dies kann an verbleibenden Concurrency Bugs liegen, die noch nicht eliminiert werden konnten. Dabei hängt sich das OSGi Framework beim Start auf. In den meisten Fällen kann einfach WOD neu gestartet werden. Im schlimmsten Fall muss es über den Task Manager oder **kill** abgeschossen werden. Startet es anschliessend immer noch nicht, kann das `cache` Verzeichnis im Daten-Verzeichnis gelöscht werden. Damit initialisiert sich das OSGi Framework beim nächsten Start komplett neu.

1.6.2. Fehlermeldungen beim Zugriff auf Dateien über UNC Pfade

Falls beim Zugriff auf Dateien über UNC Pfade Fehlermeldungen wie "Connection refused" gefunden werden, könnte es sein, dass eine veraltete Version von WOD im Einsatz ist. Das sieht man u.a. daran, dass im Stacktrace die Klasse "FtpURLConnection" erwähnt wird. In diesem Fall ist WOD zu aktualisieren.

Kapitel 2. Konfiguration

Dieses Kapitel beschreibt die Konfiguration von World Of Documents.

2.1. Übersicht

Die meisten Konfigurationsdateien befinden sich im `etc/` Verzeichnis. Üblicherweise sind es sogenannte Property-Dateien (eine Liste von Namen/Werte-Paaren) oder XML-Dateien.

Weitere Konfigurationen (z.B. Job Definitionen) können über das Deployment in das System eingebracht werden.

2.2. System Properties

Die Datei `etc/system.properties` enthält Properties, welche direkt innerhalb Java als sogenannte System Properties zur Verfügung stehen.

Hier kann z.B. das "app.data" Verzeichnis, welches den Ablage-Ort von Daten steuert, spezifiziert werden. Hier wird beispielsweise der Start Level für das OSGi Framework festgelegt.

Das "app.log" System Property kann gesetzt werden um das Log Verzeichnis woanders als im Datenverzeichnis abzulegen (z.B. `/var/log/wod` unter Unix). Standardmässig zeigt "app.log" auf "${app.data}/log".

2.3. OSGi Framework Properties

Die Datei `etc/framework.properties` enthält Properties, welche vom OSGi Framework benützt werden. Hier befindet sich u.a. auch die Konfiguration für das `etc/` Verzeichnis, welche von Apache Felix FileInstall[1] verwendet wird (felix.fileinstall.*).

Weitere Informationen über die unterstützten Properties kann in der Dokumentation zum Apache Felix Framework[2] nachgelesen werden.

`framework.properties` referenziert intern noch andere Properties Dateien, die üblicherweise nicht angepasst werden müssen:

- `endorsed.properties`: Java Packages, die über das `lib/endorsed` Verzeichnis Java-eigene Bibliotheken überschreibt.
- `jre.properties`: Java Packages, die über das OSGi Framework zur Verfügung gestellt werden.

2.4. Logging

WOD verwendet standardmässig Pax Logging verwendet, welches unter der Haube Log4J 1.2.x[3] verwendet. Die Konfigurationsdatei `etc/org.ops4j.pax.logging.cfg` enthält die Log4J 1.2 Konfiguration im Property File Format. Diese Datei kann zur Laufzeit verändert werden. Änderungen werden innert Sekunden aktiv geschalten, da Felix FileInstall sämtliche Konfigurationen (*.cfg und *.xml) im `etc/` Verzeichnis überwacht.

2.5. TCP/IP Ports

WOD stellt verschiedene TCP/IP Ports zur Verfügung.

[1] http://felix.apache.org/documentation/subprojects/apache-felix-file-install.html
[2] http://felix.apache.org/documentation/subprojects/apache-felix-framework/apache-felix-framework-configuration-properties.html
[3] http://logging.apache.org/log4j/1.2/

Tabelle 2.1. WOD Ports

Beschreibung	Standard Port Nummer
LPD (LPR Druck Server, optional)	515
SSH Shell	40022
HTTP Server	40080
RMI Server	40098
RMI Registry	40099
HTTPS Server (optional)	40043

Neben diesen Ports können noch zusätzliche aktiv sein. Diese können z.B. von der JVM selbst zur Verfügung gestellt worden sein (z.B. für JMX Zugriff) oder durch spezielle Zusatzkomponenten für WOD (beispielsweise durch ein Message Queue Server). Jede Komponente hat ihre Eigenheiten bei der Konfiguration.

Die obigen Ports im 400xx Bereich können als Gruppe angepasst werden, um z.B. mehrere Versionen von WOD parallel auf einer Maschine zu installieren. Dazu kann in der Datei `<WOD-HOME>/etc/system.properties` der Wert "wod.base.port" angepasst werden. Standardmässig ist "400" eingestellt, was beispielsweise den HTTP Port auf "40080" einstellt. Wird dieser Wert auf "410" gestellt, wird der HTTP Port "41080". Dasselbe für die anderen Ports im "400xx" Bereich in obiger Liste.

2.5.1. HTTP Konfiguration

Die Administration von WOD erfolgt primär vom Browser aus. Weiter bietet es diverse HTTP REST basierte APIs. Dazu betreibt WOD einen internen Web Server (typischerweise basierend auf Jetty). Der voreingestellte HTTP Port ist 40080 (siehe oben). Dies lässt sich aber nach Bedarf anpassen. Die Datei `org.apache.felix.http.cfg` enthält die Konfiguration für den HTTP Service.

Leider muss unter Umständen auch die Datei `etc/framework.properties` angepasst werden, da der HTTP Service aufgrund der Einstellungen dort schon den HTTP Server hoch zieht, bevor die OSGi Konfiguration in Aktion tritt. Da der hart-kodierte Default Port 8080 ist, kann es ohne Anpassungen hier Port-Kollisionen mit anderen HTTP Servern auf derselben Maschine führen.

Detailierte Information über die möglichen Einstellungen finden sich in der Dokumentation zum Apache Felix HTTP Service[4]. Dies beinhaltet auch die Aktivierung von HTTPS Unterstützung.

2.5.2. RMI

RMI steht für "Remote Method Invocation" und ist ein Standard-Mechanismus von Java für entfernte Methoden-Aufrufe. Dies wird von WOD zur Zeit nicht verwendet, ist jedoch ein notwendiger Bestandteil für den Zugriff auf JMX (Java Management Extensions). Dieser Teil kann in den meisten Fällen ignoriert werden.

Die Konfiguration `org.apache.karaf.management.cfg` enthält die Konfiguration der beiden RMI Ports (Standard: Server 40098 und Registry 40099). In der Konfiguration können sowohl die Hostnamen wie auch die Ports angepasst werden.

Der JMX Layer (inklusive RMI Konfiguration) wird von Apache Karaf[5] Komponenten zur Verfügung gestellt. Dessen Dokumentation liefert zusätzliche Informationen zur Konfiguration gestellt.

[4] http://felix.apache.org/documentation/subprojects/apache-felix-http-service.html
[5] http://karaf.apache.org/manual/latest/users-guide/monitoring.html

2.5.3. SSH Shell

WOD bietet eine Shell/Konsole[6] zur textuellen Interaktion mit dem System. Auf diese kann in erster Linie über SSH zugegriffen werden. Die Konfigurationsdatei für diese Komponente ist `org.apache.karaf.shell.cfg`. Dort können sowohl der Hostname als auch der Port angepasst werden.

Der SSH Server wird von Apache Karaf[7] Komponenten zur Verfügung gestellt. Dessen Dokumentation liefert zusätzliche Informationen zur Konfiguration gestellt.

2.6. Datenbank

WOD speichert u.a. das Audit Log (Job Protokoll) in einer Datenbank, so dass im Nachhinein sämtliche abgearbeiteten Jobs eingesehen werden können, was insbesondere bei Problemen nützlich ist. Weiter gibt es eine Variante des Content Repository, welches seinen Index in einer Datenbank speichert. In der WOD Standard Edition ist nach der Installation die Embedded Datenbank Apache Derby als Haupt-Datenbank konfiguriert. Das reicht für kleinere Installationen im Normalfall aus.

Zur Zeit unterstützt WOD als Datenbank *Apache Derby* und *PostgresQL*. Auf Anfrage können auch weitere Datenbanken unterstützt werden.

Apache Derby reicht als Datenbank aus für Installationen mit weniger als ein paar Tausend Jobs pro Tag. Geht die Anzahl der täglichen Jobs in die Zehntausende, wird angeraten z.B. auf *PostgresQL* umzustellen.

2.6.1. Apache Derby

Apache Derby ist eine in Java geschriebene Embedded Datenbank unter der Apache Lizenz Version 2.0. Sie ist die standardmässig aktivierte und konfigurierte Datenbank für WOD (Standard Edition). Die eingebettete (embedded) Datenbank hat den Vorteil, dass nicht noch eine separate Datenbank installiert und konfiguriert werden muss. Allerdings ist sie weniger für grössere Mengen geeignet.

Das Datenverzeichnis für *Apache Derby* in WOD liegt unter dessen `data` Verzeichnis im Unterverzeichnis "derby". Dies kann bei Bedarf über die Konfiguration angepasst werden.

Beispiel 2.1. Standard-Konfiguration für die Job Datenbank mit Derby (Dateiname: <wod-home>/etc/ch.jm.db.jdbc.provider.derby.DerbyEmbeddedService-WODJobDatabase.cfg)

```
name=WODJobDatabase
database-name=${app.data}/derby/wodjobdb
create-database=create
```

Siehe auch Abschnitt 2.7.2, „JM :: Apache Derby Data Source (embedded mode)“.

Dies erzeugt eine Datenquelle namens "WODJobDatabase" mit *Apache Derby*. Falls die Datenbank noch nicht existiert, wird sie erstellt. Damit ist erst die Datenquelle erzeugt. Es muss jetzt noch sichergestellt werden, dass die Datenbank als Job Datenbank verwendet werden kann. Dazu dient folgende Konfiguration:

Beispiel 2.2. Standard-Konfiguration für die Benützung von Derby als Job Datenbank (Dateiname: <wod-home>/etc/ch.jm.wod.job.model.cayenne.PersistenceProvider.cfg)

```
DATASOURCE.target = (name=WODJobDatabase)
```

Siehe auch Abschnitt 2.7.37, „WOD :: Job :: Data Model (Cayenne)“.

"WODJobDatabase" ist der Name der Datenbank, wie er oben im "name" Property gesetzt wurde. Über diesen Namen werden die zwei Komponenten verknüpft.

[6] http://karaf.apache.org/manual/latest/users-guide/console.html
[7] http://karaf.apache.org/manual/latest/users-guide/remote.html

Als nächstes wird im Normalfall der Index für das Content Repository benötigt.

Beispiel 2.3. Standard-Konfiguration für den Content Repository Index mit Derby (Dateiname: <wod-home>/etc/ch.jm.db.jdbc.provider.derby.DerbyEmbeddedService-WODContentRepo.cfg)

```
name=WODContentRepo
database-name=${app.data}/derby/wod-repo
create-database=create
```

Siehe auch Abschnitt 2.7.2, „JM :: Apache Derby Data Source (embedded mode)“.

Das Content Repository speichert seine Dokument normalerweise auf dem lokalen Dateisystem. Aber für einen schnellen Zugriff werden gewisse Informationen indexiert, und dieser Index wird standardmässig ebenfalls in *Apache Derby* gespeichert. Diese Index-Komponente ist in der Standard-Installation folgendermassen konfiguriert:

Beispiel 2.4. Standard-Konfiguration für den Content Repository Index (Dateiname: <wod-home>/etc/ch.jm.wod.repository.content.filesystem.index.xmldb-default.cfg)

```
name=default
collection-uri=rdb:WODContentRepo?ContentRepoMetadata
```

Siehe auch Abschnitt 2.7.45, „WOD :: Content Repository :: Filesystem-based Implementation :: XML:DB-based index“.

Hier wird die Verbindung der Datenquelle mit der Index-Komponente ebenfalls über dessen Namen (Property: name) gemacht (hier: "WODContentRepo"). Die "collection-uri" benützt eine URI in der Form: "rdb:<Datenquellenname>?<Datenmodellname>".

2.6.2. PostgresQL

Sollen grössere Mengen an Jobs pro Tag (z.B. mehr als 10'000), wird angeraten von *Apache Derby* auf *PostgresQL* umzusteigen. Dazu muss *PostgresQL* separat installiert werden. Informationen dazu sind der PostgresQL Website[8] zu entnehmen.

Um nun auf PostgresQL als Datenbank umzusteigen, müssen analog zu Derby oben, die entsprechenden Datenquellen konfiguriert werden. Nachfolgend die zwei Datenquellen als direkter Ersatz der Standard-Datenquellen auf Basis von Derby wie in Abschnitt 2.6.1, „Apache Derby“ beschrieben.

Beispiel 2.5. Standard-Konfiguration für die Job Datenbank mit PostgresQL (Dateiname: <wod-home>/etc/ch.jm.db.jdbc.provider.postgresql.PostgresQLService-WODJobDatabase.cfg)

```
name=WODJobDatabase
connect-string=jdbc:postgresql://127.0.0.1:5433/WODJobDatabase
uid=wod
pwd=password
max-active=32
min-idle = 0

watchdog.retry-interval=PT30S
watchdog.invalid-count-before-unregistering=2
```

Siehe auch Abschnitt 2.7.3, „JM :: PostgresQL DataSource“.

Dieses Beispiel verbindet sich auf eine PostgresQL Instanz auf derselben Maschine, erlaubt maximal 32 aktive Verbindungen und hat einen Watchdog Service, welcher alle 30 Sekunden überprüft, ob die Datenbank immer noch erreichbar ist.

Bei der Umstellung von Derby auf PostgresQL muss die entsprechende Konfiguration der Derby Datenquelle entfernt werden, da sonst zwei Datenquellen mit demselben Namen im System aktiv sind, so dass nicht mehr unbedingt gewährleistet ist, dass jeweils die richtige Datenquelle angesprochen wird.

[8] http://www.postgresql.org

Hier noch das entsprechende Beispiel für den Content Repository Index:

Beispiel 2.6. Standard-Konfiguration für die Content Repository Index mit PostgresQL (Dateiname: <wod-home>/etc/ch.jm.db.jdbc.provider.postgresql.PostgresQLService-WODContentRepo.cfg)

```
name=WODContentRepo
connect-string=jdbc:postgresql://127.0.0.1:5433/WODContentRepo
uid=wod
pwd=password

watchdog.retry-interval=PT30S
watchdog.invalid-count-before-unregistering=2
```

Siehe auch Abschnitt 2.7.3, „JM :: PostgresQL DataSource“.

Die beiden Datenbanken werden nicht automatisch erstellt. Dies muss vorher manuell geschehen, und der gewählte Username (uid) muss die entsprechenden Berechtigungen haben, so dass WOD die notwendigen Tabellen beim ersten Start erstellen kann.

2.7. Beschreibung der Konfigurations-Einheiten

2.7.1. JM :: CIFS :: Authenticator

Typ: "Factory" Konfiguration, PID: ch.jm.cifs.auth.simple

Beispiel-Dateinamen:
`ch.jm.cifs.auth.simple-Config1.cfg`
`ch.jm.cifs.auth.simple-MeinService.cfg`

Name	Beschreibung
hosts+	Hosts Typ: *String* An array of host names that apply for the login defined here.
domain?	Domain Typ: *String*, optional The Windows domain (optional)
username	Username Typ: *String* The username
password	Password Typ: *String* The password (clear text!!!)

2.7.2. JM :: Apache Derby Data Source (embedded mode)

Typ: "Factory" Konfiguration, PID: ch.jm.db.jdbc.provider.derby.DerbyEmbeddedService

Beispiel-Dateinamen:
`ch.jm.db.jdbc.provider.derby.DerbyEmbeddedService-Config1.cfg`
`ch.jm.db.jdbc.provider.derby.DerbyEmbeddedService-MeinService.cfg`

Name	Beschreibung
name	Name Typ: *String* The name of the data source.
database-name	Database Name

Name	Beschreibung
	Typ: *String* This property must be set and it identifies which database to access. Example: "G:/db/wombat"
uid	Username Typ: *String* The user name.
pwd	Password Typ: *String* The password.
create-database	Create database? Typ: *String* If set to the string "create", this will cause a new database of databaseName if that database does not already exist. The database is created when a connection object is obtained from the data source.

2.7.3. JM :: PostgresQL DataSource

Typ: "Factory" Konfiguration, PID: ch.jm.db.jdbc.provider.postgresql.PostgresQLService

Beispiel-Dateinamen:
`ch.jm.db.jdbc.provider.postgresql.PostgresQLService-Config1.cfg`
`ch.jm.db.jdbc.provider.postgresql.PostgresQLService-MeinService.cfg`

Name	Beschreibung
name	Name Typ: *String* The name of the data source.
connect-string	Connect String Typ: *String* The connect string for the PostgresQL database, for example: jdbc:postgresql://localhost/MyDatabase
uid	Username Typ: *String* The user name.
pwd	Password Typ: *String* The password.
max-active	Maximum Active Connections Typ: *Integer* The maximum number of active connections, or negative value for no limit. Default: 8.
min-idle	Minimum Idle Connections Typ: *Integer* The minimum number of connections that can remain idle in the pool, without extra ones being created, or zero to create none. Default: 0.
max-idle	Maximum Idle Connections Typ: *Integer* The maximum number of connections that can remain idle in the pool, without extra ones being released, or negative for no limit. Default: 8.

Name	Beschreibung
watchdog.invalid-count-before-unregistering?	Watchdog: Invalid Count Before Unregistering Typ: *Integer*, optional The number of invalid state checks before the service is unregistered.
watchdog.retry-interval?	Watchdog: Retry Interval Typ: *String*, optional The interval (ISO 8601 format, ex. "PT5S") in which service validity is checked.

2.7.4. JM :: E-Mail :: Sender :: SMTP

Typ: "Factory" Konfiguration, PID: ch.jm.email.sender.smtp

Beispiel-Dateinamen:
`ch.jm.email.sender.smtp-Config1.cfg`
`ch.jm.email.sender.smtp-MeinService.cfg`

E-Mail sender service using the SMTP or SMTPS protocol to send e-mail messages.

Name	Beschreibung
name?	Service Name Typ: *String*, optional The name of the e-mail sender service.
service.ranking?	Service Ranking Typ: *String*, optional The OSGi service ranking (advanced setting).
retry-for?	Retry For Typ: *String*, optional ISO 8601 duration indicating for how long sending the e-mail should be retried in case of an unavailable SMTP server. Default: PT5M
minimum-wait-between-retries?	Minimum Wait Between Retries Typ: *String*, optional ISO 8601 duration indicating who long to wait at least until retrying to send the e-mail in case of an unavailable SMTP server. Default: PT30S
maximum-wait-between-retries?	Maximum Wait Between Retries Typ: *String*, optional ISO 8601 duration indicating who long to wait at most until retrying to send the e-mail in case of an unavailable SMTP server. Default: PT5M
mail.smtp.host	SMTP Host Typ: *String* The hostname or IP address of the SMTP host.
mail.smtp.user?	SMTP Username Typ: *String*, optional The username for the SMTP server if authentication is necessary.
mail.smtp.password?	SMTP Password Typ: *String*, optional The password for the SMTP server if authentication is necessary.
mail.smtp.port?	SMTP Port

Name	Beschreibung
	Typ: *Integer*, optional The port for the SMTP server. Default: 25
mail.smtp.auth?	Authentication enabled? Typ: *Boolean*, optional If true, attempt to authenticate the user using the AUTH command. Default: false
mail.smtp.starttls.enable?	STARTTLS Support Enabled? Typ: *Boolean*, optional If true, enables the use of the STARTTLS command (if supported by the server) to switch the connection to a TLS-protected connection before issuing any login commands. Default: false
mail.smtp.starttls.required?	STARTTLS Support Required? Typ: *Boolean*, optional If true, requires the use of the STARTTLS command. If the server doesn't support the STARTTLS command, or the command fails, the connect method will fail. Default: false

2.7.5. JM :: E-Mail :: Sender :: File System

Typ: "Factory" Konfiguration, PID: ch.jm.email.sender.filesystem

Beispiel-Dateinamen:
`ch.jm.email.sender.filesystem-Config1.cfg`
`ch.jm.email.sender.filesystem-MeinService.cfg`

E-Mail sender service writing e-mail messages to the local file system.

Name	Beschreibung
name?	Service Name Typ: *String*, optional The name of the e-mail sender service.
service.ranking?	Service Ranking Typ: *String*, optional The OSGi service ranking (advanced setting).
target-directory	Target Directory Typ: *String* The target directory into which to write the fax messages.

2.7.6. JM :: Fax :: Client :: eCall E-Mail to Fax

Typ: "Factory" Konfiguration, PID: ch.jm.fax.client.ecall.email2fax

Beispiel-Dateinamen:
`ch.jm.fax.client.ecall.email2fax-Config1.cfg`
`ch.jm.fax.client.ecall.email2fax-MeinService.cfg`

Fax client service sending fax messages via the eCall e-mail to fax bridge.

Name	Beschreibung
name	Service Name Typ: *String* The name of the fax client service.
service.ranking?	Service Ranking

Name	Beschreibung
	Typ: *Integer*, optional The OSGi service ranking (advanced setting).
mail-service?	E-Mail Service Name Typ: *String*, optional The name of the e-mail sender service to use for sending fax messages via eCall. (optional)
from	Sender E-Mail Address Typ: *String* The sender e-mail address that is enabled as a valid sender in the eCall configuration.
answer-to?	Target E-Mail Address for delivery status Typ: *String*, optional The e-mail address that delivery status notifications are sent to. (optional)
bcc?	Blind Carbon Copy E-Mail Address Typ: *String*, optional The e-mail address to which all outgoing e-mails for fax delivery are sent for archival purposes. (optional)
calling-number?	Calling Fax Number Typ: *String*, optional The calling fax number which designates the sender of the fax message. (optional)
fax-header-id?	Fax Header ID Typ: *String*, optional The identification of the fax message sender shown in the header line of the fax. (optional, max. 20 characters)
fax-header-info?	Fax Header Info Typ: *String*, optional The header line on the fax message. (optional, max. 50 characters)
max-retries?	Maximum Number of Retries Typ: *Integer*, Gültig: 0-5, optional The number of times eCall™ tries to send the fax message if the receiver is already occupied or is temporarily unavailable. (optional, valid: 0..5)
retries-time-interval-in-min?	Time Between Retries Typ: *Integer*, Gültig: 0-99, optional The time in minutes between retry attempts. (optional, valid: 0..99)

2.7.7. JM :: Fax :: Client :: File-system

Typ: "Factory" Konfiguration, PID: ch.jm.fax.client.filesystem

Beispiel-Dateinamen:
`ch.jm.fax.client.filesystem-Config1.cfg`
`ch.jm.fax.client.filesystem-MeinService.cfg`

Name	Beschreibung
name	Service Name Typ: *String* The name of the fax client service.

Name	Beschreibung
target-directory	Target Directory Typ: *String* The target directory into which to write the fax messages.
service.ranking?	Service Ranking Typ: *String*, optional The OSGi service ranking (advanced setting).

2.7.8. JM :: Apache FOP OSGi Integration

Typ: "Factory" Konfiguration, PID: ch.jm.fop.osgi.FopFactoryFactory

Beispiel-Dateinamen:
`ch.jm.fop.osgi.FopFactoryFactory-Config1.cfg`
`ch.jm.fop.osgi.FopFactoryFactory-MeinService.cfg`

Name	Beschreibung
userconfig-uri	Cofiguration File Typ: *String* An optional URI to the FOP configuration file (userconfig.xml).
base-uri	Base URI Typ: *String* An optional base URI used for URI resolution.

2.7.9. JM :: OSGi :: Events :: JMS Adapter

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.events.jms.impl.GeneralEventAdapter

Beispiel-Dateinamen:
`ch.jm.osgi.events.jms.impl.GeneralEventAdapter-Config1.cfg`
`ch.jm.osgi.events.jms.impl.GeneralEventAdapter-MeinService.cfg`

Adapter which forwards OSGi events to a JMS topic or queue.

Name	Beschreibung
event.topics+	OSGi event topics Typ: *String* Lists the OSGi event topics that will be handled. Example: "ch/jm/wod/job/Signal/SUCCESS"
event.filter?	OSGi event filter Typ: *String*, optional The optional filter (LDAP syntax) to further restrict the events that will be handled. Example: "(job-type=http://example.ch/test-job)"
destination?	JMS destination Typ: *String*, optional Standardwert: ch.jm.osgi.events The name of JMS destination to send the message to.
destination.type?	JMS destination type Typ: *String*, optional Standardwert: topic Optionen: topic (topic), queue (queue) The JMS destination type. Either "topic" or "queue".

Name	Beschreibung
ttl?	Message TTL Typ: *String*, optional Standardwert: P1D The time to live for the message being sent. This value is specified as an ISO 8601 duration. The default is "P1D" (one day). Specify "P0D" to disable the TTL.
send.timeout?	Message send timeout Typ: *String*, optional Standardwert: PT5M Specifies the timeout that is used when sending a JMS message. This value is specified as an ISO 8601 duration. Default: "PT5M" (five minutes)

2.7.10. JM :: OSGi :: Health Check :: Servlet

Typ: Einfache Konfiguration, PID: ch.jm.osgi.healthcheck.servlet.HealthCheckServlet

Dateiname:
`ch.jm.osgi.healthcheck.servlet.HealthCheckServlet.cfg`

The health check servlet for monitoring the OSGi system.

Name	Beschreibung
alias	Servlet Alias Typ: *String* Standardwert: /health The alias (mount point) for the servlet, ex. "/health"

2.7.11. JM :: OSGi :: HTTP :: Debug Filter

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.http.filter.impl.DebugFilter

Beispiel-Dateinamen:
`ch.jm.osgi.http.filter.impl.DebugFilter-Config1.cfg`
`ch.jm.osgi.http.filter.impl.DebugFilter-MeinService.cfg`

Servlet Filter logging information on HTTP requests.

Name	Beschreibung
pattern	Pattern Typ: *String* Regular expression pattern to register filter with.
service.ranking	Service Ranking Typ: *String* The service ranking. Influences in which position in the filter chain this instance will be placed.
contextId	Context ID Typ: *String* The context ID for the HttpContext.

2.7.12. JM :: OSGi :: HTTP :: Error Filter

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.http.filter.impl.ErrorFilter

Beispiel-Dateinamen:

`ch.jm.osgi.http.filter.impl.ErrorFilter-Config1.cfg`
`ch.jm.osgi.http.filter.impl.ErrorFilter-MeinService.cfg`

Servlet Filter logging problems on HTTP requests.

Name	Beschreibung
pattern	Pattern Typ: *String* Regular expression pattern to register filter with.
service.ranking	Service Ranking Typ: *String* The service ranking. Influences in which position in the filter chain this instance will be placed.
contextId	Context ID Typ: *String* The context ID for the HttpContext.

2.7.13. JM :: OSGi :: HTTP :: Basic Authentication :: HttpContext

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.http.security.impl.BasicAuthHttpContext

Beispiel-Dateinamen:
`ch.jm.osgi.http.security.impl.BasicAuthHttpContext-Config1.cfg`
`ch.jm.osgi.http.security.impl.BasicAuthHttpContext-MeinService.cfg`

HttpContext services for HTTP Basic Authentication.

Name	Beschreibung
realm	Realm Typ: *String* The authentication realm for the service.
contextId	Context ID Typ: *String* The context ID for the HttpContext.
roles.allowed+	Allowed Roles Typ: *String* A list of roles that are allowed to access the associated resources.
secure.connection.required?	Secure Connection Required? Typ: *Boolean*, optional Standardwert: false Indicates whether a secure (HTTPS) connection is required for access to the associated resources.

2.7.14. JM :: OSGi :: HTTP :: Basic Authentication :: Servlet Filter

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.http.security.impl.BasicAuthFilter

Beispiel-Dateinamen:
`ch.jm.osgi.http.security.impl.BasicAuthFilter-Config1.cfg`
`ch.jm.osgi.http.security.impl.BasicAuthFilter-MeinService.cfg`

Servlet Filter services for HTTP Basic Authentication.

Name	Beschreibung
realm	Realm

Name	Beschreibung
	Typ: *String* The authentication realm for the service.
pattern	Pattern Typ: *String* Regular expression pattern to register filter with.
service.ranking	Service Ranking Typ: *String* The service ranking. Influences in which position in the filter chain this instance will be placed.
contextId	Context ID Typ: *String* The context ID for the HttpContext.
roles.allowed+	Allowed Roles Typ: *String* A list of roles that are allowed to access the associated resources.
secure.connection.required?	Secure Connection Required? Typ: *Boolean*, optional Standardwert: false Indicates whether a secure (HTTPS) connection is required for access to the associated resources.

2.7.15. JM :: OSGi :: HTTP :: Loopback Authentication Filter

Typ: Einfache Konfiguration, PID: ch.jm.osgi.http.security.impl.LoopbackAuthFilter

Dateiname:
`ch.jm.osgi.http.security.impl.LoopbackAuthFilter.cfg`

Servlet Filter for authenticating connections on the loopback interface.

Name	Beschreibung
username	Username Typ: *String* The username under which to authenticate connections on the loopback interface.
pattern	Pattern Typ: *String* Regular expression pattern to register filter with.
service.ranking	Service Ranking Typ: *String* The service ranking. Influences in which position in the filter chain this instance will be placed.
contextId	Context ID Typ: *String* The context ID for the HttpContext.

2.7.16. JM :: OSGi :: HTTP :: Rule-based Authentication Filter

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.http.security.impl.HeaderRuleAuthFilter

Beispiel-Dateinamen:
`ch.jm.osgi.http.security.impl.HeaderRuleAuthFilter-Config1.cfg`
`ch.jm.osgi.http.security.impl.HeaderRuleAuthFilter-MeinService.cfg`

Servlet Filter for authenticating connections using rules applied to HTTP headers.

Name	Beschreibung
username	Username Typ: *String* The username under which to authenticate connections on the loopback interface.
pattern	Pattern Typ: *String* Regular expression pattern to register filter with.
service.ranking	Service Ranking Typ: *String* The service ranking. Influences in which position in the filter chain this instance will be placed.
contextId	Context ID Typ: *String* The context ID for the HttpContext.
hosts.allowed+	Allowed Hosts Typ: *String* A list of hostname or IP addresses to allowed hosts. ("*" for all hosts)
headers+	%header.name Typ: *String* An optional list of allowed headers in the format "<Header-Name>: <regex>", ex. "User-Agent: Java/.*".

2.7.17. JM :: OSGi :: HTTP :: Certificate Pre-Authentication Filter

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.http.security.impl.CertificatePreAuthFilter

Beispiel-Dateinamen:
`ch.jm.osgi.http.security.impl.CertificatePreAuthFilter-Config1.cfg`
`ch.jm.osgi.http.security.impl.CertificatePreAuthFilter-MeinService.cfg`

Servlet Filter for for preparing certificate authentication.

Name	Beschreibung
pattern	Pattern Typ: *String* Regular expression pattern to register filter with.
service.ranking	Service Ranking Typ: *String* The service ranking. Influences in which position in the filter chain this instance will be placed.
contextId	Context ID Typ: *String* The context ID for the HttpContext.

2.7.18. JM :: Installer :: Artifact Synchronization :: XML-based Artifact Repository

Typ: Einfache Konfiguration, PID: ch.jm.osgi.installer.sync.xml.XMLBasedProfileRepository

Dateiname:

`ch.jm.osgi.installer.sync.xml.XMLBasedProfileRepository.cfg`

Name	Beschreibung
location	Repository XML location Typ: *String* The location of the Repository XML. The value must be a valid URL.

2.7.19. JM :: Installer :: Artifact Synchronization :: Provider Servlet

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.installer.sync.servlet

Beispiel-Dateinamen:
`ch.jm.osgi.installer.sync.servlet-Config1.cfg`
`ch.jm.osgi.installer.sync.servlet-MeinService.cfg`

Name	Beschreibung
root	Root directory Typ: *String* The root directory for the artifact profile repository. The value must be a valid and existing directory path.
interval?	Interval Typ: *Long*, Gültig: 0-, optional The frequency at which to check the monitored directory for changes. The value is a number and denotes milliseconds.
alias?	Servlet Alias Typ: *String*, optional The alias at which the servlet is deployed. Example: /artifacts

2.7.20. JM :: Installer :: Artifact Transformer

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.installer.transformer

Beispiel-Dateinamen:
`ch.jm.osgi.installer.transformer-Config1.cfg`
`ch.jm.osgi.installer.transformer-MeinService.cfg`

Name	Beschreibung
root	Root directory Typ: *String* The root directory for the source files. The value must be a valid and existing directory path.
target-directory	Target directory Typ: *String* The directory where the generated artifacts are written to. The value must be a valid and existing directory path.
interval?	Interval Typ: *Long*, Gültig: 1-, optional Standardwert: 5000 The frequency at which to check the monitored root directory for changes. The value is a positive number and denotes milliseconds. By default, the root directory is checked every 5 seconds.

2.7.21. JM :: Pax Logging Configurator

Typ: Einfache Konfiguration, PID: ch.jm.osgi.logging.paxhelper

Dateiname:
`ch.jm.osgi.logging.paxhelper.cfg`

Name	Beschreibung
file	Log4J Configuration File Typ: *String* The absolute path of the Log4J configuration file (usually called log4j.properties).
interval	Check Interval Typ: *String* Interval (in seconds) in which to check for changes in the configuration file

2.7.22. JM :: OSGi :: Net :: Authenticator

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.net.authenticator.impl.simple

Beispiel-Dateinamen:
`ch.jm.osgi.net.authenticator.impl.simple-Config1.cfg`
`ch.jm.osgi.net.authenticator.impl.simple-MeinService.cfg`

Name	Beschreibung
protocols+	%protocols.name Typ: *String* %protocols.desc
hosts+	Hosts Typ: *String* An array of host names that apply for the login defined here.
username	Username Typ: *String* The username
password	Password Typ: *String* The password (clear text!!!)

2.7.23. JM :: OSGi :: OrientDB Graph Source

Typ: "Factory" Konfiguration, PID: ch.jm.osgi.orientdb.client.OrientGraphSource

Beispiel-Dateinamen:
`ch.jm.osgi.orientdb.client.OrientGraphSource-Config1.cfg`
`ch.jm.osgi.orientdb.client.OrientGraphSource-MeinService.cfg`

Source service for OrientDB Graph databases.

Name	Beschreibung
name	Name Typ: *String* The name of the source.
url	Connection URL Typ: *String*

Name	Beschreibung
	The connection URL to the OrientDB database. Ex. "remote:srv01/test"
username?	Username Typ: *String*, optional The username with which to connect to the database.
password?	Password Typ: *String*, optional The password with which to connect to the database.
min-connections?	Minimum Connections Typ: *Integer*, Gültig: 0-1024, optional Standardwert: 0 The minimum number of connection to keep in the connection pool.
max-connections?	Maximum Connections Typ: *Integer*, Gültig: 0-1024, optional Standardwert: 0 The maximum number of connection to keep in the connection pool. Use 0 to disable the pool and create a new connection whenever a connection is requested.
watchdog.invalid-count-before-unregistering?	Watchdog: Invalid Count Before Unregistering Typ: *Integer*, optional The number of invalid state checks before the service is unregistered.
watchdog.retry-interval?	Watchdog: Retry Interval Typ: *String*, optional The interval (ISO 8601 format, ex. "PT5S") in which service validity is checked.

2.7.24. JM :: OSGi Remote Services Implementation (using Hessian)

Typ: Einfache Konfiguration, PID: ch.jm.osgi.remote.exports

Dateiname:
`ch.jm.osgi.remote.exports.cfg`

Name	Beschreibung
service-base-url	Service Base URL Typ: *String* The base URL used when constructing external service URLs. (There's currently no way to automatically determine to correct host name. This defaults to "http://localhost:{org.osgi.service.http.port}/".

2.7.25. JM :: PEAX Client Account

Typ: "Factory" Konfiguration, PID: ch.jm.peax.client.Account

Beispiel-Dateinamen:
`ch.jm.peax.client.Account-Config1.cfg`
`ch.jm.peax.client.Account-MeinService.cfg`

Configuration for a PEAX.ch client account.

Name	Beschreibung
name	Account Name Typ: *String* The name for the PEAX client account which will be used to look up the client.
api.base-url	API base URL Typ: *String* The base URL for the PEAX API (v1), ex. "https://api.peax.ch/api/v1/"
oauth.base-url	OAuth base URL Typ: *String* The base URL for the OAuth endpoint used for authentication, ex. "https://login.peax.ch/sts/connect/token"
oauth.client-id	OAuth client ID Typ: *String* The client ID to use for OAuth authentication
oauth.secret	OAuth secret Typ: *String* The secret to use for OAuth authentication (has obfuscation support)
oauth.scope	OAuth scope Typ: *String* Standardwert: delivery_create The scope to use for OAuth authentication. Default: "delivery_create"

2.7.26. JM :: Pingen Account

Typ: "Factory" Konfiguration, PID: ch.jm.pingen.Account

Beispiel-Dateinamen:
`ch.jm.pingen.Account-Config1.cfg`
`ch.jm.pingen.Account-MeinService.cfg`

Configuration for a Pingen.com account.

Name	Beschreibung
name	Account Name Typ: *String* The name for the Pingen account which will be used to look up the client. Usually the e-mail address for the account.
environment	Target environment Typ: *String* Standardwert: production Optionen: production (Production Environment), staging (Staging Environment) The target environment
api.token	API Token Typ: *String* The API token used for authentication
default.color?	Default Color Setting Typ: *String*, optional Optionen: grayscale (Grayscale), color (Color), mixed (Mixed/Optimized Color) The default color type used (optional). Example: mixed

Name	Beschreibung
default.duplex?	Default Duplex mode Typ: *String*, optional Optionen: simplex (Simplex), duplex (Duplex) The default duplex mode (optional). Example: simplex
default.address-placement?	Default Address Placement Typ: *String*, optional Optionen: left (Left), right (Right) The default address placement (optional). Example: right
default.speed?	Default Speed Typ: *Integer*, optional The default delivery speed (optional). Example: 1 (for priority), 2 (for economy)

2.7.27. WOD :: Delivery :: File

Typ: "Factory" Konfiguration, PID: ch.jm.wod.delivery.file

Beispiel-Dateinamen:
`ch.jm.wod.delivery.file-Config1.cfg`
`ch.jm.wod.delivery.file-MeinService.cfg`

Name	Beschreibung
target.directory	Target Directory Typ: *String* The target directory in which to place delivered files.

2.7.28. WOD :: Delivery :: LPR

Typ: "Factory" Konfiguration, PID: ch.jm.wod.delivery.lpr

Beispiel-Dateinamen:
`ch.jm.wod.delivery.lpr-Config1.cfg`
`ch.jm.wod.delivery.lpr-MeinService.cfg`

Name	Beschreibung
hostname	LPR Server Typ: *String* The host name of the LPR server.
queue	Queue Typ: *String* The name of the printer queue to deliver the document to.

2.7.29. WOD :: Delivery :: SFTP

Typ: "Factory" Konfiguration, PID: ch.jm.wod.delivery.sftp

Beispiel-Dateinamen:
`ch.jm.wod.delivery.sftp-Config1.cfg`
`ch.jm.wod.delivery.sftp-MeinService.cfg`

Name	Beschreibung
host	SFTP Host

Name	Beschreibung
	Typ: *String* The hostname for the SFTP server.
port	SFTP Port Typ: *Integer* The SFTP port. Default: 22
username	Username Typ: *String* The user name.
password	Password Typ: *String* The password.
targetpath	Target Path Typ: *String* The target path in which to place delivered files.
knownHosts	Known Hosts File Typ: *String* The URI pointing to the file with known hosts. This file is expected to have the same format as OpenSSH's known_hosts file. It is used for authenticating known servers.

2.7.30. WOD :: e-Bill :: SIX Paynet biller account

Typ: "Factory" Konfiguration, PID: ch.jm.wod.ebill.six.BillerAccount

Beispiel-Dateinamen:
`ch.jm.wod.ebill.six.BillerAccount-Config1.cfg`
`ch.jm.wod.ebill.six.BillerAccount-MeinService.cfg`

Configuration for a SIX Paynet biller account.

Name	Beschreibung
biller.id	Biller Account ID Typ: *String* The biller account ID assigned by SIX. Example: 41010100000000000
biller.name?	Biller Account Name Typ: *String*, optional The biller account name (optional, informational only)
target	Target Web Service Typ: *String* Standardwert: test Optionen: test (Test Environment), prod (Production Environment), custom (Custom Environment) The target Web Service (KI or Prod) that shall be used. May be empty.
endpoint?	Endpoint Typ: *String*, optional The custom HTTP URL for the DWS service if none of the standard URLs are used.
username	Username Typ: *String* The username for access to the web service.
password	Password

Name	Beschreibung
	Typ: *String* The (optionally obfuscated) password for access to the web service when using password authentication.
tenant?	Tenant Typ: *String*, optional Indicates the (optional) tenant that shall be used when creating a job for a downloaded document.
download.scheduler.expression?	Shipment Download Task Schedule Typ: *String*, optional The optional cron-syntax schedule for the shipment download task.
processing-report.job-type?	Shipment Processing Report Job Type Typ: *String*, optional The job type URI to use for the jobs created for shipment processing reports. (optional)

2.7.31. WOD :: e-Bill :: SIX Paynet downloaded document handler :: Job Submission

Typ: "Factory" Konfiguration, PID: ch.jm.wod.ebill.six.JobSubmittingDocumentHandler

Beispiel-Dateinamen:
`ch.jm.wod.ebill.six.JobSubmittingDocumentHandler-Config1.cfg`
`ch.jm.wod.ebill.six.JobSubmittingDocumentHandler-MeinService.cfg`

This service submits WOD jobs for processing downloaded documents.

Name	Beschreibung
biller.id?	Biller Account ID Typ: *String*, optional The biller account ID assigned by SIX. Example: 41010100000000000
biller.name?	Biller Account Name Typ: *String*, optional The biller account name (optional, informational only)
shipment.type*	Shipment Type URI Typ: *String*, optional Restricts the applicability of this handler to certain shipment formats (such as "six:/paynet/Structured/Bansta.EIXML.2003A.BCONS15").
job-type	Job Type for processing the shipment (downloaded document) Typ: *String* The job type URI to use for the jobs created when downloading a shipment.
service.ranking?	Service Ranking Typ: *String*, optional The OSGi service ranking (advanced setting).

2.7.32. WOD :: e-Bill :: yellowbill biller account

Typ: "Factory" Konfiguration, PID: ch.jm.wod.ebill.yellowbill.BillerAccount

Beispiel-Dateinamen:

```
ch.jm.wod.ebill.yellowbill.BillerAccount-Config1.cfg
ch.jm.wod.ebill.yellowbill.BillerAccount-MeinService.cfg
```

Name	Beschreibung
biller.id	Biller Account ID Typ: *String* The biller account ID assigned by PostFinance. Example: 41101000000322421
biller.name?	Biller Account Name Typ: *String*, optional The biller account name (optional, informational only)
target	Target Web Service Typ: *String* Standardwert: ki Optionen: ki (Customer Integration Environment), prod (Production Environment), custom (Custom Environment) The target Web Service (KI or Prod) that shall be used. May be empty.
endpoint.upload?	Upload Endpoint Typ: *String*, optional The custom HTTP URL for the Upload service if none of the standard URLs are used.
endpoint.biller-download?	Biller Download Endpoint Typ: *String*, optional The custom HTTP URL for the Biller Download service if none of the standard URLs are used.
username	Username Typ: *String* The username for access to the web service.
password	Password Typ: *String* The (optionally obfuscated) password for access to the web service when using password authentication.
tenant?	Tenant Typ: *String*, optional Indicates the (optional) tenant that shall be used when creating a job for a downloaded signed invoice.
download.job-type	Invoice Download Job Type Typ: *String* The job type URI to use for the jobs created when downloading a signed invoice.
download.scheduler.expression?	Invoice Download Task Schedule Typ: *String*, optional The optional cron-syntax schedule for the invoice download task.
process-protocol.job-type?	Process Protocol Download Job Type Typ: *String*, optional The job type URI to use for the jobs created for downloading process protocols. (optional)
process-protocol.scheduler.expression?	Process Protocol Task Schedule Typ: *String*, optional The optional cron-syntax schedule for the process protocols download task.

2.7.33. WOD :: Job :: Controller :: Distributed

Typ: Einfache Konfiguration, PID: ch.jm.wod.job.controller.hazelcast.pull.JobController

Dateiname:
`ch.jm.wod.job.controller.hazelcast.pull.JobController.cfg`

Name	Beschreibung
hazelcast?	Hazelcast Instance Name Typ: *String*, optional The name of the Hazelcast instance to use (may be empty for the default instance).
worker-count?	Worker Count Typ: *Integer*, optional The number of workers (threads) to run for handling jobs. (Optional, defaults to the number of CPUs + 1)

2.7.34. WOD :: Job :: Controller :: In-Memory Implementation

Typ: Einfache Konfiguration, PID: ch.jm.wod.job.controller.inmemory.impl.JobHandler

Dateiname:
`ch.jm.wod.job.controller.inmemory.impl.JobHandler.cfg`

Name	Beschreibung
thread-pool-size	Thread Pool Size Typ: *Integer* The size of the regular thread pool for job handlers.
max-low-priority-jobs	Maximum Low-Priority Jobs Typ: *String* The number of maximum, concurrent, low-priority jobs.

2.7.35. WOD :: Job :: Events :: E-Mail Adapter

Typ: "Factory" Konfiguration, PID: ch.jm.wod.job.events.mail.EMailAdapter

Beispiel-Dateinamen:
`ch.jm.wod.job.events.mail.EMailAdapter-Config1.cfg`
`ch.jm.wod.job.events.mail.EMailAdapter-MeinService.cfg`

Name	Beschreibung
smtp.host	SMTP Host Typ: *String* The SMTP host to use for sending e-mail. Default: localhost
smtp.username	SMTP Username Typ: *String* The username to use for sending e-mail via SMTP.
smtp.password	SMTP Password Typ: *String* The password to use for sending e-mail via SMTP.
email.from	Sender Address (From) Typ: *String*

Name	Beschreibung
	Specifies the sender address of the e-mail.
email.to	Receiver (To) Typ: *String* Specifies the receiver of the e-mail.
event.topics+	Event Topics Typ: *String* A list of Event Topics to listen to. Example: ch/jm/wod/job/Signal/FAILURE

2.7.36. WOD :: Job :: Events :: XMPP Adapter

Typ: "Factory" Konfiguration, PID: ch.jm.wod.job.events.xmpp.XMPPAdapter

Beispiel-Dateinamen:
`ch.jm.wod.job.events.xmpp.XMPPAdapter-Config1.cfg`
`ch.jm.wod.job.events.xmpp.XMPPAdapter-MeinService.cfg`

Name	Beschreibung
xmpp.host	XMPP Host Typ: *String* The XMPP host to connect to. Example: Default: talk.google.com
xmpp.port?	XMPP Port Typ: *Integer*, optional The XMPP server port to use. Default: 5222
xmpp.user	XMPP Username Typ: *String* The username of the XMPP account to use.
xmpp.domain	XMPP Domain Typ: *String* The domain of the XMPP account.
xmpp.password	XMPP Password Typ: *String* The password of the XMPP account to use.
recipients+	Recipients Typ: *String* Specifies a list of recipients for the event message.
event.topics+	Event Topics Typ: *String* A list of Event Topics to listen to. Example: ch/jm/wod/job/Signal/FAILURE

2.7.37. WOD :: Job :: Data Model (Cayenne)

Typ: Einfache Konfiguration, PID: ch.jm.wod.job.model.cayenne.PersistenceProvider

Dateiname:
`ch.jm.wod.job.model.cayenne.PersistenceProvider.cfg`

Configuration for a Cayenne-based persistence service.

Name	Beschreibung
datasource	Data Source Name

Name	Beschreibung
	Typ: *String* The name of the data source.

2.7.38. WOD :: Job :: Submission :: REST Interface

Typ: Einfache Konfiguration, PID: ch.jm.wod.job.submission.rest

Dateiname:
`ch.jm.wod.job.submission.rest.cfg`

Name	Beschreibung
alias	ContextId Typ: *String* The context ID under which to publish the REST interface, ex. "wod".
contextId	%contextId.name Typ: *String* %contextId.desc

2.7.39. WOD :: Job :: Submission

Typ: Einfache Konfiguration, PID: ch.jm.wod.job.submission

Dateiname:
`ch.jm.wod.job.submission.cfg`

Name	Beschreibung
default-job-expiration	Default Job Expiration Typ: *String* The duration in which an job expires. The format is "PnYnMnDTnHnMnS", as defined in XML Schema 1.0 section 3.2.6.1

2.7.40. WOD :: Job :: Submission :: LPR/LPD Queue

Typ: "Factory" Konfiguration, PID: ch.jm.wod.job.submission.lpr

Beispiel-Dateinamen:
`ch.jm.wod.job.submission.lpr-Config1.cfg`
`ch.jm.wod.job.submission.lpr-MeinService.cfg`

Name	Beschreibung
queue-name	Tenant Typ: *String* The tenant to assign the submitted jobs (optional).
tenant?	%tenant.name Typ: *String*, optional %tenant.desc
mime-type	MIME type (optional) Typ: *String* An optional MIME type to skip job identification stage or to handle a potentially ambiguous case. Example: application/pdf

Name	Beschreibung
job-type	Job Type URI (optional) Typ: *String* An optional Job Type URI to skip job identification stage or to handle a potentially ambiguous case.

2.7.41. WOD :: Job :: Submission :: Runnable Task

Typ: "Factory" Konfiguration, PID: ch.jm.wod.job.submission.runnable

Beispiel-Dateinamen:
`ch.jm.wod.job.submission.runnable-Config1.cfg`
`ch.jm.wod.job.submission.runnable-MeinService.cfg`

Name	Beschreibung
tenant?	Tenant (optional) Typ: *String*, optional An optional tenant to assign the job to.
job-type	Job Type URI (optional) Typ: *String* An optional Job Type URI to skip job identification stage or to handle a potentially ambiguous case.
mime-type	MIME type (optional) Typ: *String* An optional MIME type to skip job identification stage or to handle a potentially ambiguous case. Example: application/pdf
payload-source	Payload Source (URI, optional) Typ: *String* A URI indicating a fixed source for a job payload.
scheduler.expression	Schedule (optional) Typ: *String* A CRON-type scheduler expression for scheduling regular execution of the task.

2.7.42. WOD :: Content Repository :: Maintenance Job :: Remove Expired Documents

Typ: Einfache Konfiguration, PID: ch.jm.wod.repository.content.impl.RemoveExpiredDocumentsJob

Dateiname:
`ch.jm.wod.repository.content.impl.RemoveExpiredDocumentsJob.cfg`

Configuration for a maintenance job (Runnable) that removes expired documents from the content repository.

Name	Beschreibung
scheduler.expression	Scheduler Expression Typ: *String* A scheduler expression as supported by Quartz (used by Apache Sling Scheduler), ex. "0 0 20 ? * *" which runs the job every day at 20:00. See: http://sling.apache.org/site/scheduler-service-commons-scheduler.html

2.7.43. WOD :: Content Repository :: CMIS-based Implementation

Typ: "Factory" Konfiguration, PID: ch.jm.wod.repository.content.cmis

Beispiel-Dateinamen:
`ch.jm.wod.repository.content.cmis-Config1.cfg`
`ch.jm.wod.repository.content.cmis-MeinService.cfg`

Name	Beschreibung
name	Content Repository Name Typ: *String* The name of the content repository.
root-folder	Root Folder Typ: *String* The root folder inside the CMIS repository for the content repository. Examples: "/" or "/wod"
document-type	CMIS Document Type Typ: *String* The CMIS document type that is used when creating the documents. Example: "File" or "cmis:document" (the default)
expiration-property	Expiration Property Name Typ: *String* The name of the property in which to store the expiration value for the document (if set). Example: "wod:expires" (the default) or "dc:expired" (used by Nuxeo with document type "File")
org.apache.chemistry.opencmis.sessic	Repository ID Typ: *String* The ID of the repository to open. Example: "default"
org.apache.chemistry.opencmis.bindir	AtomPub URL Typ: *String* The URL for AtomPub access to the CMIS-capable document repository. Example: "http://localhost:8080/nuxeo/atom/cmis"
org.apache.chemistry.opencmis.user	Username Typ: *String* The username for access to the document repository
org.apache.chemistry.opencmis.passw	Password Typ: *String* The password for access to the document repository (WARNING: this is stored in clear text!)

2.7.44. WOD :: Content Repository :: Filesystem-based Implementation

Typ: Einfache Konfiguration, PID: ch.jm.wod.repository.content.filesystem

Dateiname:
`ch.jm.wod.repository.content.filesystem.cfg`

Filesystem-based implementation of the simple content repository API.

Name	Beschreibung
root	Root Directory

Name	Beschreibung
	Typ: *String* The root directory on the local file system for data storage.
database	XML:DB Database Name Typ: *String* Standardwert: rdb The name of the XML:DB database to use. Default: "rdb". This setting is deprecated. Set an index name instead.
collection-uri	XML:DB Collection URI Typ: *String* The name of the XML:DB database to use. Default: "rdb". This setting is deprecated. Set an index name instead.
index	Index Name Typ: *String* The name of the index.

2.7.45. WOD :: Content Repository :: Filesystem-based Implementation :: XML:DB-based index

Typ: "Factory" Konfiguration, PID: ch.jm.wod.repository.content.filesystem.index.xmldb

Beispiel-Dateinamen:
`ch.jm.wod.repository.content.filesystem.index.xmldb-Config1.cfg`
`ch.jm.wod.repository.content.filesystem.index.xmldb-MeinService.cfg`

The XML:DB-based index for the simple content repository API.

Name	Beschreibung
name	Index Name Typ: *String* The name of the index.
database	XML:DB Database Name Typ: *String* Standardwert: rdb The name of the XML:DB database to use. Default: "rdb".
collection-uri	XML:DB Collection URI Typ: *String* The XML:DB collection URI used for storing metadata and indices.

2.7.46. WOD :: Content Repository :: Filesystem-based Implementation :: Solr-based index

Typ: "Factory" Konfiguration, PID: ch.jm.wod.repository.content.filesystem.index.solr

Beispiel-Dateinamen:
`ch.jm.wod.repository.content.filesystem.index.solr-Config1.cfg`
`ch.jm.wod.repository.content.filesystem.index.solr-MeinService.cfg`

The Solr/Lucene-based index for the simple content repository API.

Name	Beschreibung
name	Index Name Typ: *String*

Name	Beschreibung
	The name of the index.
core-name	Solr core name Typ: *String* The name of the Solr core to use as the index backend. Example: "main/repo"

2.7.47. WOD :: Content Repository :: REST Interface

Typ: Einfache Konfiguration, PID: ch.jm.wod.repository.content.rest

Dateiname:
`ch.jm.wod.repository.content.rest.cfg`

Name	Beschreibung
alias	Alias Typ: *String* Resource path under which to publish the REST interface, ex. "/wod-repo".

2.7.48. WOD :: Workflow :: Remote Camunda BPM Engine

Typ: "Factory" Konfiguration, PID: ch.jm.wod.workflow.camunda.http.engine

Beispiel-Dateinamen:
`ch.jm.wod.workflow.camunda.http.engine-Config1.cfg`
`ch.jm.wod.workflow.camunda.http.engine-MeinService.cfg`

A remote Camunda BPM engine.

Name	Beschreibung
name	Service Name Typ: *String* The name for the Camunda BPM engine.
base-uri	REST engine base URL Typ: *String* The base URL of the Camunda engine's REST interface.

Kapitel 3. Schnittstellen

Dieses Kapitel behandelt die externen Schnittstellen, die World Of Documents zur Verfügung stellt. Darüber können z.B. Jobs gestartet und deren Status abgefragt werden.

3.1. HTTP/REST Schnittstelle

Diese Schnittstelle bedient hauptsächlich zwei Anforderungen: die Initiierung von Jobs und deren Überwachung bzw. Manipulation. Es werden zwei Möglichkeiten zum Start von Jobs bereitgestellt: die asynchrone Verarbeitung, z.B. für grössere Jobs, und die synchrone Verarbeitung, hauptsächlich für Einzeldokumente, wo das Resultat des Jobs direkt als Antwort auf den Aufruf zurückgegeben wird.

3.1.1. Asynchrone Verarbeitung

Hierbei geht es um den Start von Jobs, wobei die Beendigung des Jobs irgendwann in der Zukunft stattfindet. Der Aufrufer erhält als Resultat des Aufrufs eine eindeutige ID für den Job, mit dem er anschliessend den Status aktiv abrufen kann oder passiv auf ein Beendigungs-Signal warten kann. Dies wird hauptsächlich für grössere (Batch-)Jobs verwendet, oder wenn Einzeldokumente über eine gewisse Zeit gesammelt und dann zusammen verarbeitet werden sollen.

Die relative URL für die Schnittstelle lautet: `/wod/jobs/submit`. Die vollständige URL könnte damit also z.B. lauten: `http://wod-server:40080/wod/jobs/submit`. Es wird ein HTTP POST verwendet.

Als Antwort auf den Request wird HTTP Status 201 (CREATED) zurückgeliefert. Der HTTP Header "Location" enthält dabei die (Host-relative) URL des Jobs, über die z.B. Status-Informationen abgefragt werden können. Als Payload wird ein XML Dokument zurückgeliefert, welcher folgendermassen aussehen kann (hier inkl. der gelieferten HTTP Header, die nicht Teil des XML Dokuments sind):

```
< HTTP/1.1 201 Created
< Date: Fri, 07 Jun 2013 08:46:33 GMT
< Server: Jetty(7.x.y-SNAPSHOT)
< Location: https://localhost:40080/wod/job/8480
< Content-Type: application/xml
< Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<resource xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns="http://jeremias-maerki.ch/wod"
    id="8480"
    xlink:href="/wod/job/8480">
<uri>wod:default:job:8480</uri>
</resource>
```

3.1.2. Synchrone Verarbeitung

Hierbei geht es um die direkt, synchrone Produktion von Dokumenten. Das Resultat des Jobs wird direkt als Payload der Antwort auf den HTTP Request zurück gegeben. Das ist nur für Einzeldokumente sinnvoll, welche innerhalb von wenigen Sekunden erzeugt werden können. Der aufgerufene Job muss dabei ein einzelnes Dokument erzeugen, damit dies funktionieren kann. Solche Dokumente werden standardmässig als Jobs mit Priorität 10 erzeugt, werden also in jedem Fall direkt in die Verarbeitung genommen. Kann das Dokument nicht innerhalb von 45 Sekunden erzeugt werden, z.B. bei hoher System-Auslastung, wird der HTTP Status 201 (CREATED) statt 200 (OK) zurückgegeben. In diesem Fall ist das Resultat dasselbe, als wäre die asynchrone Verarbeitung angestossen worden (siehe Abschnitt 3.1.1, „Asynchrone Verarbeitung“). Schlägt der Job fehl, wird ein entsprechender HTTP Fehler Code zurückgegeben.

Die relative URL für die Schnittstelle lautet: `/wod/jobs/produceDocument`. Die vollständige URL könnte damit also z.B. lauten: `http://wod-server:40080/wod/jobs/produceDocument`. Es wird ein HTTP POST verwendet.

3.1.3. Der HTTP Payload

Der Payload für HTTP Requests kann in 3 Varianten erfolgen. Dies gilt für synchrone wie asynchrone Aufrufe.

3.1.3.1. application/x-www-form-urlencoded

Diese Methode entspricht dem, was standardmässig von einem HTML-Formular über HTTP POST übermittelt wird. Das genaue Format des Payload ist der HTML 4 Spezifikation[1] zu entnehmen.

Tabelle 3.1. Verfügbare HTML Formular-Felder

Name	Beschreibung
uri	Die URI oder URL des Inhaltes des zu verarbeitenden Dokuments.
job-type	(optional) Die URI des Job Typs, falls bekannt. Die Angabe dieses Parameters ermöglicht das Überspringen der automatischen Ermittlung des Inhaltstyps und verbessert damit die Performance.

Ein solcher Aufruf kann z.B. auch über das Kommandozeilen-Tool **curl** erzeugt werden. Hier ein Beispiel:

```
curl
  --data-urlencode "uri=file:/var/opt/myapp/test.xml"
  --header "X-WOD-JobType:http://test.com/invoice"
  --header "X-WOD-invoicenr:1234"
  -X POST http://localhost:40080/wod/jobs/submit
```

Über diesen Aufruf kann nicht direkt ein Eingabe-Dokument übergeben werden, sondern lediglich eine URI/URL darauf. Sollen direkt Dokumenteninhalte (z.B. XML Daten) an WOD übergeben werden, muss eine der folgenden Methoden verwendet werden.

3.1.3.2. multipart/form-data

Diese Methode entspricht dem, was benützt wird, wenn mit einem HTML-Formular eine Datei hochgeladen werden soll. Dabei wird auf dem HTML-Formular das Attribut `enctype="multipart/form-data"` gesetzt. Das genaue Format des Payload ist der HTML 4 Spezifikation[2] zu entnehmen. Diese Schnittstelle verarbeitet ausschliesslich den ersten Part des Multipart Paketes. Werden also mehrere Dateien gleichzeitig übermittelt, wird lediglich die erste übernommmen. Der im Part angegebene Media Type (z.B. "application/xml") wird für die Ermittlung des Job Typs bei Bedarf herangezogen, um die Performance zu verbessern. Ist dieser ungekannt, soll "application/octet-stream" verwendet werden.

Andere Felder als der erste sogenannte Body Part mit der hochgeladenen Datei werden zur Zeit ignoriert. Zusätzliche Parameter sollen über HTTP Header (siehe Abschnitt 3.1.4, „Allgemein unterstützte HTTP POST Header“) übertragen werden.

3.1.3.3. Direkter Upload

Bei dieser Methode wird das hochzuladende Dokument direkt als Payload des HTTP POST Requests übertragen. Der im HTTP Request angegebene Media Type (z.B. "application/xml") wird für die Ermittlung des Job Typs bei Bedarf herangezogen, um die Performance zu verbessern. Ist dieser ungekannt, soll "application/octet-stream" verwendet werden. Zusätzliche Parameter können über HTTP Header (siehe Abschnitt 3.1.4, „Allgemein unterstützte HTTP POST Header“) übertragen werden.

Ein solcher Aufruf kann z.B. auch über das Kommandozeilen-Tool **curl** erzeugt werden. Hier ein Beispiel:

```
curl
    --upload-file /var/opt/myapp/test.xml
    --header "X-WOD-JobType:http://test.com/invoice"
    --header "X-WOD-invoicenr:1234"
    -X POST http://localhost:40080/wod/jobs/submit
```

[1] http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.1
[2] http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.2

3.1.4. Allgemein unterstützte HTTP POST Header

Beide HTTP POST Aufrufarten unterstützen zusätzliche Parameter über HTTP Header Einträge.

Tabelle 3.2. Verfügbare HTTP Header

Name	Beschreibung
X-WOD-JobType	(Typ: URI) Die URI des Job Typs, falls bekannt. Die Angabe dieses Parameters ermöglicht das Überspringen der automatischen Ermittlung des Inhaltstyps und verbessert damit die Performance.
X-WOD-JobPriority	(Standard: 0, Typ: Ganzzahl) Definiert die initiale Job Priorität. Alle Werte über 0 bezeichnen prioritäre Jobs, die so schnell wie möglich ausgeführt werden. Nicht-prioritäre Jobs werden u.U. zeitverzögert ausgeführt.
X-WOD-Tenant	(Typ: String) Gibt den Mandanten an, für den der Job gestartet werden soll. Der Header kann weggelassen werden, wenn nicht mit Mandanten gearbeitet wird.
X-WOD-*	Alle nicht speziell dokumentierten HTTP Header, die mit "X-WOD-" anfangen, werden als Job Attribute behandelt. Beispielsweise führt "X-WOD-Rechnungsnummer: 8762348" zu einem Job-Attribut namens "Rechnungsnummer", welches den Wert "8762348" enthalten wird.

3.1.5. Job-Abfrage

Über HTTP lassen sich auch Jobs bzw. Listen von Jobs abfragen, sowie eine einfache Statistik der aktuellen Jobs. Sämtliche GET URLs können auch aus dem Browser abgerufen werden und werden in diesem Fall in formatiert in HTML angezeigt.

GET http://localhost:40080/wod/job/{id}

Retourniert Informationen über einen bestimmten Job.
Pfad Parameter: {id} = Job ID
Rückgabeformate: application/xhtml+xml, text/xml, application/xml
cURL Beispiel: `curl -v -H "Accept: text/xml" http://localhost:40080/wod/job/2941`

Beispiel 3.1. Job XML Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<job xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://jeremias-maerki.ch/wod"
      id="2941" xlink:href="/wod/job/2941">
  <uri>wod:default:job:2941</uri>
  <type>job-type:pdf2print</type>
  <status>CLOSED</status>
  <created>2015-09-25T15:49:13.374+02:00</created>
  <expires>2015-10-25T15:49:13.373+01:00</expires>
  <priority>0</priority>
  <attributes>
    <printer-name>HP LaserJet Pro CM1415fn color MFP</printer-name>
    <incoming-channel xmlns="http://www.jeremias-maerki.ch/ns/wod/job"
        >http:submit</incoming-channel>
    <fixed>test</fixed>
  </attributes>
  <events>
    <trace created="2015-09-25T15:49:13.396+02:00">
      <message code="STATUS_CHANGE">Job status changed to PENDING after 22ms</message>
    </trace>
    <result created="2015-09-25T15:49:14.923+02:00">
      <message code="MAIN_PRODUCT">Main product: wod:default:content:2015/09/[..]7137.ps
        (ContentType:application/postscript, application/postscript)</message>
    </result>
    <trace created="2015-09-25T15:49:14.929+02:00">
      <message code="LOG">Log output</message>
      <details><![CDATA[[INFO] ch.jm.wod.pipeline.storage.pipes.StorePipe$1: Stored[..]
```

```
[..]
[INFO] Document printed to file:///D:/out.pcl.
  drop-off-point: file:///D:/out.pcl
  thread: ByteStreamAdapterPipe-34-thread-1
  format: ContentType:application/postscript
  size: 1083366
]]></details>
    </trace>
    <trace created="2015-09-25T15:49:14.929+02:00">
      <message code="PERF">Pipeline universal-print-pdf executed in 1516ms.</message>
    </trace>
    <trace created="2015-09-25T15:49:14.930+02:00">
      <message code="STATUS_CHANGE">Job status changed to CLOSED after 1555ms</message>
    </trace>
  </events>
  <main-document xlink:href="/wod/document/2921">
    <document id="2921" xlink:href="/wod/document/2921">
      <representations>
        <representation id="3707" xlink:href="/wod/representation/3707">
          <uri xlink:href="/wod/representation/3707"
              >wod:default:representation:3707</uri>
          <type>ContentType:application/postscript</type>
          <mime>application/postscript</mime>
          <content-uri xlink:href="http://localhost:40080/wod-repo/document/[..]37.ps"
            >wod:default:content:2015/09/25/15/7097d110-505b-4[..]37.ps</content-uri>
        </representation>
        <representation id="3706" xlink:href="/wod/representation/3706">
          <uri xlink:href="/wod/representation/3706"
              >wod:default:representation:3706</uri>
          <type>http://www.jeremias-maerki.ch/ns/document/structure</type>
          <mime>text/xml</mime>
          <content-uri xlink:href="http://localhost:40080/wod-repo/document/[..]6.xml"
            >wod:default:content:2015/09/25/15/a84688cb-4ed2-4[..]6.xml</content-uri>
        </representation>
        <representation id="3705" xlink:href="/wod/representation/3705">
          <uri xlink:href="/wod/representation/3705"
              >wod:default:representation:3705</uri>
          <type>http://www.jeremias-maerki.ch/ns/document/content</type>
          <mime>text/xml</mime>
          <content-uri xlink:href="http://localhost:40080/wod-repo/document/[..]5.xml"
            >wod:default:content:2015/09/25/15/e8768fa4-225c-4[..]5.xml</content-uri>
        </representation>
        <representation id="3704" xlink:href="/wod/representation/3704">
          <uri xlink:href="/wod/representation/3704"
              >wod:default:representation:3704</uri>
          <type>ContentType:application/pdf</type>
          <mime>application/pdf</mime>
          <content-uri xlink:href="http://localhost:40080/wod-repo/document/[..]6.pdf"
            >wod:default:content:2015/09/25/15/a9d645bd-80d7-4[..]6.pdf</content-uri>
        </representation>
      </representations>
    </document>
  </main-document>
</job>
```

GET http://localhost:40080/wod/job/{id}/status

Retourniert den aktuelle Status eines Jobs.
Pfad Parameter: {id} = Job ID
Rückgabeformate: text/plain
Rückgabewerte: NEW, HELD, PENDING, FAILED, CANCELLED, CLOSED
cURL Beispiel: `curl -v http://localhost:40080/wod/job/2941/status`

PUT/POST http://localhost:40080/wod/job/{id}/status

Setzt den aktuellen Status eines Jobs. Gültige Statuswerte hier sind "NEW" (zum Zurücksetzen), "HELD" (um die Verarbeitung des Jobs auszusetzen) und "CANCELLED" (um den Job abzubrechen).

Parameter: {id} = Job ID
Payload: multipart/form-data (mit dem Statuswert im Parameter "status")
Rückgabeformate: n/a

GET http://localhost:40080/wod/jobs/list/

Retourniert eine Liste von Jobs.
Pfad Parameter: n/a
Query Parameter "limit": (positive integer) Die maximale Zahl von Jobs, die zurück zu liefern sind. Standardwert: 50
Query Parameter "offset": (positive integer) Der Index (0-basiert) des ersten Jobs, der zurück zu liefern ist. Standardwert: 0
Query Parameter "status": (enum) Einschränkung auf Jobs mit einem bestimmten Status. Gültige Werte: NEW, HELD, PENDING, FAILED, CANCELLED, CLOSED
Query Parameter "tenant": (string) Einschränkung auf Jobs eines bestimmten Mandanten (oder "*" für alle Mandanten)
Query Parameter "created": (date range) Einschränkung auf Jobs, die innerhalb einer bestimmten Zeitspanne erstellt wurden.
Rückgabeformate: application/xhtml+xml, text/xml, application/xml
cURL Beispiel: `curl -v "http://localhost:40080/wod/jobs/list/?offset=50&limit=10"`

Beispiele für das Format "date range" sind:

- "2010-10-21..2010-11-21": alle zwischen dem 21.10.2010 und dem 21.11.2010
- "2010-10-21..": alle seit dem 21.10.2010
- "..2010-10-21T10:00:00+02:00": alle bis am 21.10.2010 um 10:00 CEST
- "P1D": alle der letzten 24 Stunden
- "P7D..2010-10-21": alle der 7 Tage vor dem 21.10.2010

Das Format folgt dem Datums- und Dauer-Format aus XML Schema bzw. ISO 8601. Start- und Endpunkte sind immer inklusive.

Beispiel 3.2. Beispiel einer Job-Liste im XML Format

```
<?xml version="1.0" encoding="UTF-8"?>
<list xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://jeremias-maerki.ch/wod">
  <job id="3087" xlink:href="/wod/job/3087">
    <uri>wod:default:job:3087</uri>
    [..]
  </job>
  <links>
    <link rel="self" href="http://localhost:40080/wod/jobs/list?limit=10&amp;offset=50"/>
    <link rel="start" href="http://localhost:40080/wod/jobs/list?limit=10"/>
    <link rel="next" href="http://localhost:40080/wod/jobs/list?limit=10&amp;offset=60"/>
    <link rel="prev" href="http://localhost:40080/wod/jobs/list?limit=10&amp;offset=40"/>
  </links>
</list>
```

GET http://localhost:40080/wod/jobs/report/

Retourniert eine einfache Statistik über die ausgewählten Jobs.
Pfad Parameter: n/a
Query Parameter "status": (enum) Einschränkung auf Jobs mit einem bestimmten Status. Gültige Werte: NEW, HELD, PENDING, FAILED, CANCELLED, CLOSED
Query Parameter "tenant": (string) Einschränkung auf Jobs eines bestimmten Mandanten (oder "*" für alle Mandanten)
Query Parameter "created": (date range) Einschränkung auf Jobs, die innerhalb einer bestimmten Zeitspanne erstellt wurden.
Rückgabeformate: application/xhtml+xml, text/xml, application/xml, text/csv, text/comma-separated-values, text/tab-separated-values

cURL Beispiel: `curl -v -H "Accept: text/csv" "http://localhost:40080/wod/jobs/report/?status=FAILED"`

Beispiel 3.3. Beispiel eines Job-Reports im XML Format

```
<?xml version="1.0" encoding="UTF-8"?>
<table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <columns>
    <column nr="1" name="tenant"/>
    <column nr="2" name="job-type"/>
    <column nr="3" name="priority"/>
    <column nr="4" name="status"/>
    <column nr="5" name="job-count"/>
  </columns>
  <rows>
    <row nr="1">
      <tenant xsi:nil="true"/>
      <job-type>http://www.jeremias-maerki.ch/ns/demo/eurofxref</job-type>
      <priority>0</priority>
      <status>FAILED</status>
      <job-count>4</job-count>
    </row>
    <row nr="2">
      <tenant xsi:nil="true"/>
      <job-type>http://www.jeremias-maerki.ch/ns/demo/telco/batch/format</job-type>
      <priority>0</priority>
      <status>FAILED</status>
      <job-count>5</job-count>
    </row>
  </rows>
</table>
```

Beispiel 3.4. Beispiel eines Job-Reports im CSV Format

```
tenant,job-type,priority,status,job-count
"",http://www.jeremias-maerki.ch/ns/demo/eurofxref,0,FAILED
"",http://www.jeremias-maerki.ch/ns/demo/telco/batch/format,0,FAILED
```

3.2. LPD/LPR Schnittstelle

LPD (Line Printer Daemon Protokoll) ist ein älteres und einfaches Druckprotokoll, welches immer noch weit verbreitet ist. Auf der Server-Seite qird häufig der Kürzel LPD verwendet, auf der Client-Seite eher LPR (Line Printer Remote). WOD enthält u.a. einen LPD/LPR Server. Es ist möglich gleichzeitig mehrere benannte Druckerwarteschlangen (Print Queues) zu definieren, welche unterschiedliche Aktionen auslösen können (über die Job-Typ URI).

Der LPD Server wird automatisch gestartet, sobald die erste Print Queue konfiguriert ist. Anzumerken ist, dass der Port 515 nicht von einem anderen LPD Server (z.B. CUPS oder LPRng) belegt sein darf.

3.2.1. Print Queue Konfiguration

Die Konfiguration einer Print Queue geschieht über den OSGi Config Admin. In WOD wird dazu normalerweise eine *.cfg Datei angelegt, dessen Name das Muster `ch.jm.wod.job.submission.lpr-<queuename>.cfg` befolgt. Die Konfigurationsdatei wird entweder in WOD's `etc` Verzeichnis oder in ein Bundle unter dem Verzeichnis `/META-INF/cfg` abgelegt.

Beispiel 3.5. Beispiel einer LPD Queue Konfiguration (Dateiname: ch.jm.wod.job.submission.lpr-ps2pdf.cfg)

```
queue-name=ps2pdf
mime-type=application/postscript
job-type=wod:demo:ps2pdf
fixed.test=true
```

Dieses Beispiel definiert eine Print Queue namens "ps2pdf", welche PostScript als Druckformat erwartet. Sämtliche Jobs, welche über diese Print Queue hereinkommen, lösen einen Job vom Typ "wod:demo:ps2pdf" aus. Und zuletzt wird noch fix ein Job Attribut "test" mit dem Wert "true" gesetzt.

- `queue-name`: der Name der LPD Queue.
- `mime-type` (optional): der MIME Typ (z.B. application/pdf) der eingehenden Dokumente, falls der Typ des Dokuments nicht automatisch ermittelt werden soll.
- `job-type` (optional): die URI des Job Typs, der für das eigehende Dokument ausgelöst werden soll. Wird dieser Parameter weggelassen, wird der Job Typ anhand der registrierten Job Typen und deren Identifikatoren ermittelt.
- `fixed.*` (optional): Sämtliche Properties, welche mit "fixed." beginnen werden als Job Attribute mitgeführt (ohne den "fixed." Präfix).

Neben den expliziten Properties (fixed.*) setzt die LPD Queue auch noch automatisch ein paar Job Properties:

- `lpr.queue.name`: der Name der LPD Queue, über die der Job hereinkommt.
- `lpr.job.name`: der mit der LPD Session übertragen Job-Name.
- `lpr.username`: der Username des Benutzers, welcher den Druckjob auslöste.
- `lpr.source.filename`: der Dateiname des Dokumente, welches gedruckt wird.
- `lpr.hostname`: der Name (oder IP Adresse) des Hosts, welcher den Druckjob auslöste.

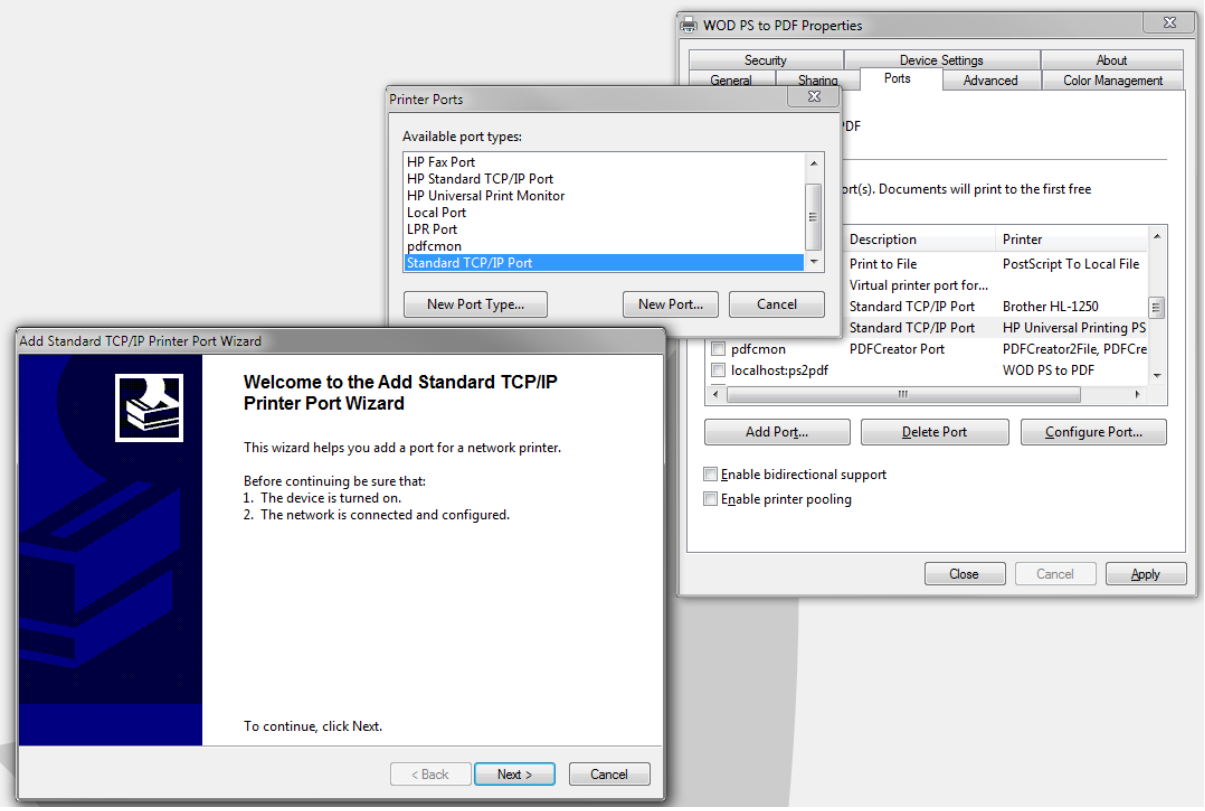
Ausser `lpr.queue.name` sind alle obigen Properties optional. Sie werden nur gesetzt, wenn die Informationen durch den LPR Client auch übergeben werden.

Das nützlichste Property wird wohl der Username sein. Hier könnte man anhand des Usernamens über einen Lookup die endgültigen Zieldrucker automatisch auswählen lassen, falls am Schluss des Workflows tatsächlich ein Dokument gedruckt wird. Oder auch eine Benachrichtigung kann so an die richtige Stelle geschickt werden.

Es wird empfohlen die Properties `mime-type` und `job-type` zu setzen. Dies beschleunigt einerseits den Start eines Jobs, weil keine Inhaltserkennung stattfinden muss. Andererseits dokumentiert es auch gleich, was eine bestimmte Queue für eine Aufgabe hat. Es kann auch sein, dass es 2 Queues gibt, welche beide PostScript entgegennehmen, aber unterschiedliche Workflows auslösen (z.B. einmal mit Archivierung und einmal ohne).

3.2.2. Einrichtung des virtuellen Druckers unter Windows

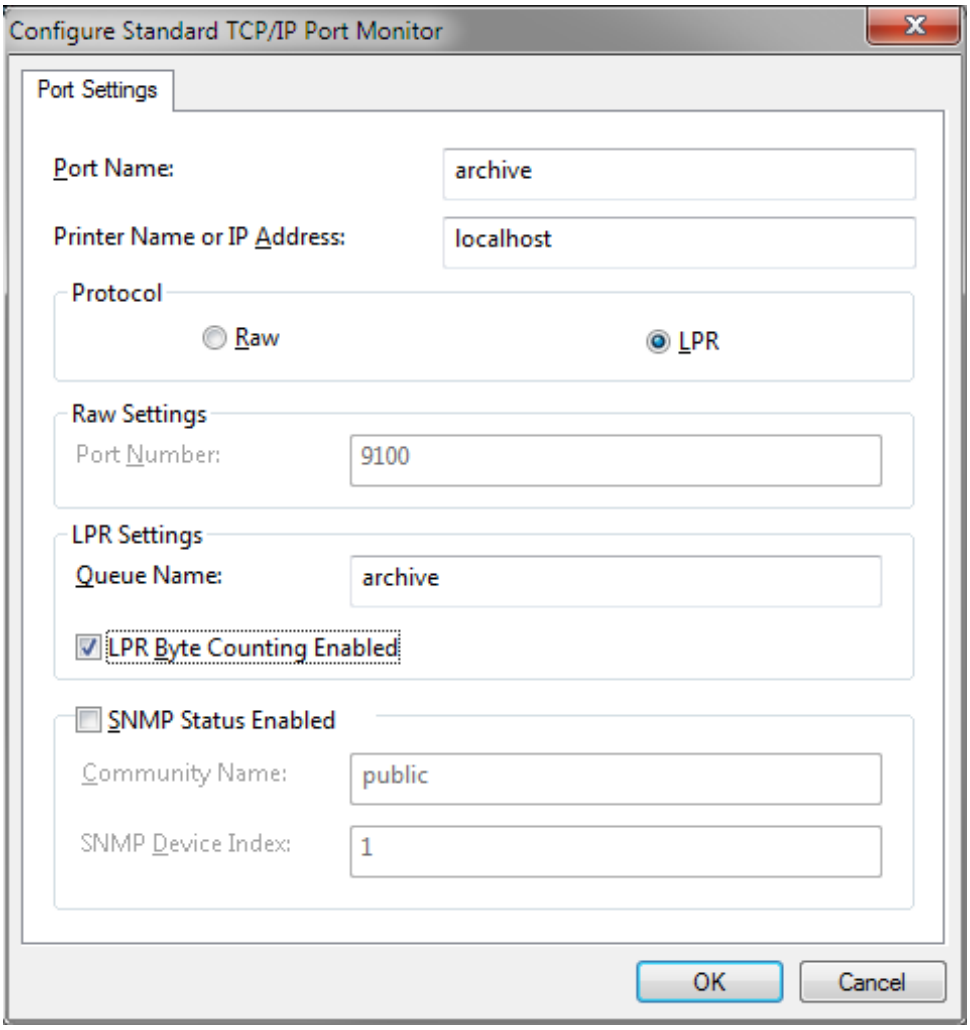
Um auf WOD als virtuellen Drucker zu drucken, kann unter Windows einfach ein PostScript-Druckertreiber installiert werden. Empfohlen wird der "HP Universal Printing PostScript" empfohlen. Da die Druckereinrichtung unter verschiedenen Windows-Version recht unterschiedlich sein kann, wird hier nur grob skizziert, was zu tun ist. Grundsätzlich ist ein Netzwerk-Drucker (Drucker mit TCP/IP Adresse oder Hostname) manuell einzurichten. Unter Umständen kann es einfacher sein, den Drucker erst an "LPR1:" oder "FILE:" einzurichten und später in den Druckereinstellungen einen neuen Port einzurichten, welches das LPR Protokoll ("Standard TCP/IP Port") unterstützt.



Der Screenshot zeigt einen Teil der Einrichtung eines LPR Ports unter Windows 7.

Der Hostname des LPD Server ist dann der Hostname der Maschine, auf der WOD läuft. Der Queue Name ist der Name, wie er in der Konfigurationsdatei (`queue-name`) angegeben ist.

In gewissen Fällen unterstützt der einzurichtende Port unter Windows gleichzeitig "Raw" (JetDirect) und "LPR". In diesem Fall muss "LPR" gewählt werden. Der Dialog sieht in etwa so aus:



Der Screenshot zeigt den Konfigurations-Dialog für
den "Standard TCP/IP Port Monitor" unter Windows 7.

3.3. Hot Folder Schnittstelle

Die Hot Folder Schnittstelle ermöglicht es ein lokales Verzeichnis zu überwachen und bei Ablage einer neuen Datei in WOD einen neuen Job auszulösen.

3.3.1. Hot Folder Konfiguration

Die Konfiguration eines Hot Folders geschieht über den OSGi Config Admin. In WOD wird dazu normalerweise eine *.cfg Datei angelegt, dessen Name das Muster `ch.jm.hotfolder.filesystem.FileSystemHotFolder-<name>.cfg` befolgt. Die Konfigurationsdatei wird entweder in WOD's `etc` Verzeichnis oder in ein Bundle unter dem Verzeichnis `/META-INF/cfg` abgelegt.

Beispiel 3.6. Beispiel einer Hot Folder Konfiguration (Dateiname: ch.jm.hotfolder.filesystem.FileSystemHotFolder-inbox.cfg)

```
LISTENER.target=(topic=wod:job)
folder.watched=D:/Temp/_hotfolder
folder.moveaway=D:/Temp/_hotfolder/_moveaway
interval=2000
minimum-idle=4000
```

Dieses Beispiel definiert das Verzeichnis `D:\Temp\_hotfolder` als Hot Folder. Alle 2 Sekunden (2000 Millisekunden) wird geprüft, ob neue Dateien aufgetaucht sind. Wird eine Datei entdeckt, wird sie erst beobachtet, und erst wenn sie sich 4 Sekunden (4000 Millisekunden) lang nicht mehr verändert hat (z.B. weil sie noch geschrieben wird), wird die Datei ins "Move-Away" Verzeichnis (hier: D:\Temp_hotfolder_moveaway) verschoben. Das stellt sicher, dass kein Programm noch einen Lock auf die Datei hat. Wenn die Datei anschliessend nicht verarbeitet werden kann, bleibt es in diesem "Move-Away" Verzeichnis. Von da kann es bei Bedarf zurück in den Hot Folder verschoben werden. Konnte die Datei verarbeitet werden, wird sie gelöscht.

Vielleicht fällt auf, dass obige Windows Pfade in der Konfiguration mit normalen Slashes (/) geschrieben wurden. Der Grund dafür: Backslashes (\) müssen in diesen Konfigurationen "escaped", also als "\\" geschrieben werden, denn der Backslash dient hier als Escape Character. Die Syntax wird hier beschrieben: java.util.Properties Text Syntax[3]

- `LISTENER.target`: dieser Wert muss fix "(topic=wod:job)" sein.
- `folder.watched`: das Verzeichnis, das überwacht werden soll.
- `folder.moveaway` (optional): das Verzeichnis, in welches neue Dateien verschiebt, bevor sie verarbeitet werden. Wird dieser Wert weggelassen, wird die Datei immer wieder in die Überwachung aufgenommen, was bedeutet, dass immer wieder versucht wird, die Datei als neuen Job aufzugeben.
- `interval` (optional): Das Interval (in Millisekunden) zwischen Prüfungen auf neue Dateien. Standardwert: 2000.
- `minimum-idle` (optional): Die Dauer, während derer die Datei sich nicht mehr verändern darf, bevor sie als neuer Job aufgegeben wird. Standardwert: 4000.

Zur Zeit ist es noch nicht möglich zusätzliche Job Properties oder direkt den Job Typ und den MIME Typ zu setzen. Dateien, die über den Hot Folder hereinkommen, werden von WOD also immer durch die automatische Job Identifikation geschickt.

[3] http://docs.oracle.com/javase/8/docs/api/java/util/Properties.html#load-java.io.Reader-

Kapitel 4. Dokumentenverarbeitung

Dieses Kapitel behandelt die verschiedenen Möglichkeiten der Dokumentenverarbeitung mit World Of Documents.

4.1. Dokumententypen

Dokument-Inhalte werden üblicherweise mit einem *Media Type* (auch *Content Type* oder *MIME Type* genannt) näher bezeichnet. PDF wird z.B. über "application/pdf" angezeigt, einfacher Text über "text/plain". Kommen wir zu XML, wird's etwas schwieriger. Der Media Type von XML ist "text/xml" (oder "application/xml"), aber XML ist nicht gleich XML. In WOD müssen wir zwischen verschiedenen XML-Formaten unterscheiden können, also brauchen wir eine genauere Beschreibung eines Dokumententyps. WOD verwendet deshalb *URIs*. Alle Media Types können in eine URI abgebildet[1] werden. Bei den Standard-Typen wird so "application/pdf" zu "ContentType:application/pdf". Bei XML können jetzt aber z.B. "mydocs:invoice" und "mydocs:delivery-note" zwei verschiedene XML-basierte Formate identifizieren. Beide haben den Media Type "text/xml". Nur über die URIs können verschiedene *Dokumententypen* also genauer unterschieden werden.

4.2. Job Definitionen

Job Definitionen sind Verarbeitungsprozesse, die über eine XML Datei definiert werden. Die Job Definition enthält Metadaten über den Job, Anweisungen für die Job-Identifikation und Verarbeitungs-Pipelines. Diese Art der Job-Definition ist momentan der empfohlene Ansatz zur Umsetzung von Dokumentenverarbeitungsprozessen.

Wie vieles in WOD wird eine Job-Definition durch eine eindeutige URI (den Job Typ) identifiziert. Zusätzlich kann eine Beschreibung und eine Versionsnummer mitgeführt werden. Hier ein Beispiel einer Job-Definition:

Beispiel 4.1. Beispiel einer Job-Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<job-definitions
  xmlns="http://www.jeremias-maerki.ch/ns/wod/job/simple"
  xmlns:job="http://www.jeremias-maerki.ch/ns/wod/job"
  xmlns:wod="http://www.jeremias-maerki.ch/ns/wod/functions">
  <job-definition>
    <job-type>job-type:ps2pdf</job-type>
    <description>PostScript to PDF Conversion</description>
    <version>1.0</version>
    <identification representation-type="ContentType:application/postscript">
      <mime>application/postscript</mime>
    </identification>
    <job-expires-in>P7D</job-expires-in>

    <execution>
      <pipeline name="ps2pdf"/>
    </execution>

    <pipelines xmlns="http://www.jeremias-maerki.ch/ns/wod/pipeline">
      <pipeline>
        <name>ps2pdf</name>
        <assembly>
          <stream source="{job:representation(job:main-document())}"
              format="ContentType:application/postscript"/>
          <exec executable="C:\gs9.02\bin\gswin32c.exe"
              output-format="application/pdf">
            <arg>-dSAFER</arg>
            <arg>-dBATCH</arg>
```

[1] http://tools.ietf.org/html/draft-eastlake-cturi-04

```
            <arg>-dNOPAUSE</arg>
            <arg>-sstdout=%stderr</arg>
            <arg>-sDEVICE=pdfwrite</arg>
            <arg>-sOutputFile=-</arg> <!-- output to stdout -->
            <arg>-</arg> <!-- input from stdin -->
          </exec>
          <store-stream expiration="PT2H"
              identifier="{wod:identifier($source-uri)}"/>
          <deliver document="{$job:generated-content}"
              target="file-target"/>
        </assembly>
      </pipeline>

    </pipelines>
  </job-definition>
</job-definitions>
```

4.2.1. Automatische Identifikation von Jobs

Bei der Job-Aufgabe kann grundsätzlich der Job-Typ explizit mitgegeben werden. Allerdings erlaubt WOD die automatische Identifikation (Auswahl) des richtigen Job Typs auf Basis des Inhalts einer ankommenden Datei. Dies kann im einfachsten Fall einfach die Ermittlung des MIME Typs (z.B. application/pdf) sein. Im Beispiel 4.1, „Beispiel einer Job-Definition“ wird der Job Typ `job-type:ps2pdf` ausgewählt, wenn eine PostScript-Datei (MIME Typ: application/postscript) empfangen wurde.

```
[..]
<identification representation-type="ContentType:application/postscript">
  <mime>application/postscript</mime>
</identification>
[..]
```

Das Attribute *representation-type* gibt eine URI (oder einen MIME Typ) an, welcher zum Speichern der empfangenen Datei verwendet werden soll.

Anmerkung zu MIME Typen

MIME Typen sind gerade im Falle von XML Dateien zu ungenau um Datei-Typen eindeutig zu identifizieren. "text/xml" ist in diesem Fall ungenügend. Eine Möglichkeit wäre für eine anwendungsspezifische XML-Datei einen privaten MIME Type wie diesen zu erstellen: application/X-myapp+xml. Allerdings verwendet WOD intern eine URI. MIME Typen werden dabei in URIs mit dem Schema "ContentType" umgewandelt: "application/postscript" wird damit zur URI "ContentType:application/postscript". An den meisten Orten wird ein MIME Typ automatisch in einer URI konvertiert.

Im Falle von XML Dateien (MIME Typen text/xml und application/xml) kann dieser Mechanismus noch verfeinert werden. Dabei ist es möglich z.B. das Root Element der XML Datei zu prüfen. Im folgenden Beispiel werden XSL-FO Dateien identifiziert:

```
[..]
<identification representation-type="http://www.w3.org/1999/XSL/Format">
  <root-element
    namespace="http://www.w3.org/1999/XSL/Format">root</root-element>
</identification>
[..]
```

Als Repräsentationstyp (anderswo auch Rendition genannt) wird hier direkt die Namespace URI von XSL-FO verwendet.

Die dritte Art der Identifikation ist ebenfalls spezifisch für XML. Hierbei wird der XML Inhalt über XPath Abfragen inspiziert, was zwar performance-mässig etwas aufwändiger ist, aber trotzdem üblicherweise recht schnell durchgeführt werden kann, wenn nicht die gesamte XML Datei durchsucht werden muss. Hier ein Beispiel mit XPath:

```
[..]
<identification
    representation-type="http://www.ecb.int/vocabulary/2002-08-01/eurofxref"
    xmlns:gesmes="http://www.gesmes.org/xml/2002-08-01">
  <xpath>/gesmes:Envelope/gesmes:subject = 'Reference rates'</xpath>
</identification>
[..]
```

In diesem Beispiel wird geschaut, ob in der vorliegenden XML Datei im "subject" Attribut der Text "Reference rates" steht. Das Beispiel zeigt auch den Umgang mit XML Namespaces.

Es gibt Situationen, wo verschiedene Dokumententypen genau gleich verarbeitet werden, so dass höchstens eine Parametrisierung des Jobs zur Laufzeit (z.B. basierend auf den Daten) geschehen muss. Diese Verarbeitung wird dann in einer einzigen Job-Definition zusammengefasst. Gibt es dann aber davon noch Spezialfälle, wird es etwas mühsam, die XPath-Queries für die Identifikation kompakt und einfach zu halten. Deshalb ist es möglich auf den `root-element` und `xpath` Elementen ein Attribut *`priority`* zu setzen, welches die Priorität der Abfrage definiert. So kann indirekt die Reihenfolge der Prüfungen beeinflusst werden. Ist das Attribut nicht gesetzt, wird die Priorität 0 verwendet. Es sind sowohl positive wie negative Werte (32bit Integer) erlaubt. In obiger Situation würde dem generischen Fall eine tiefere Priorität zugewiesen (z.B. -100). Beispiel:

```
<!-- Generischer Fall -->
<identification representation-type="acme:generic-document">
  <xpath priority="-100">boolean(/Root/Header/Report_ID)</xpath>
</identification>
```

```
<!-- Spezialfall -->
<identification representation-type="acme:special-report">
  <xpath>/Root/Header/Report_ID = 'ACME117'</xpath>
</identification>
```

So wird `acme:special-report` gewählt, wenn die "Report_ID" den Wert "ACME117" hat. Andernfalls wird `acme:generic-document` benützt. Würde hier nicht mit dem *`priority`* Attribut gearbeitet, wäre die Prüf-Reihenfolge nicht deterministisch. Die Reihenfolge hängt von der Deployment-Reihenfolge innerhalb des Systems ab, die systembedingt praktisch nicht beeinflusst werden kann.

4.2.2. Dokumententypen-Definition

Wenn WOD unbekannte MIME Typen bzw. Dokumententypen findet, muss es sinnvolle Fallbacks finden, wenn es beispielsweise eine Datei in einem Repository ablegt. Vielfach wird dabei der MIME Typ "application/octet-stream" verwendet. Eine Datei-Erweiterung gibt es so auch nicht. WOD bietet deshalb eine Möglichkeit dem System neue Dokumententypen bekannt zu machen. Dabei wird sowohl der Repräsentationstyp (eine URI), der MIME Typ, die Dateierweiterung sowie eine kurze Beschreibung angegeben. Damit hat WOD alle Informationen um eine erzeugte Datei sinnvoll zu benennen und z.B. im Browser korrekt anzuzeigen.

Hier ein Beispiel für eine Dokumententyp-Definition:

```
[..]
<document-type
  uri="http://www.jeremias-maerki.ch/ns/demo/telco"
  mime="text/xml" extension="xml"
  description="Telco Demo Invoice"/>
[..]
```

4.2.3. Weitere Einstellungen

4.2.3.1. Ablaufdatum für die Job-Instanz

Ein ausgeführter Job (Job-Instanz) wird in einer Datenbank protokolliert. Diese Informationen sind z.B. nützlich um nachträglich fehlgeschlagene Jobs analysieren zu können. Oder sie dienen der Beweissicherung, dass ein Dokument entsprechend der Vorgaben verarbeitet wurde. Je nach Fall müssen diese

Informationen allerdings nicht mehr unbedingt aufgewahrt werden. Die Einstellung `job-expires-in` erlaubt mit einer ISO 8601 Dauer anzugeben, wie lange das Job-Log aufbewahrt werden soll. "P7D" würde den Job beispielsweise 7 Tage aufbewahren und dann zur Löschung freigeben. Beispiel:

```
[..]
<job-expires-in>P7D</job-expires-in>
[..]
```

Dieser Wert muss nicht unbedingt mit der Aufbewahrungsdauer für ein Dokument in einem Archiv übereinstimmen. Das Dokument könnte 10 Jahre aufgewahrt werden, während der Job vielleicht nach einem Monat bereits gelöscht werden kann.

4.2.4. Ausführung von Jobs

Grundsätzlich sind mehrere Arten denkbar, wie ein Job ausgeführt werden kann. Im Moment gibt es zwei Möglichkeiten: ausprogrammiert in Java als OSGi-Service (zur Zeit undokumentiert), oder als Pipeline (siehe Abschnitt 4.3, „Verarbeitungs-Pipelines“).

Das `execution` Element definitiert, wie der Job ausgeführt wird. Wird eine Pipeline aufgerufen, wird das folgendermassen definiert:

```
[..]
<execution>
  <pipeline name="ps2pdf"/>
</execution>
[..]
```

In diesem Beispiel wird die Pipeline "ps2pdf" ausgeführt. Diese kann entweder direkt innerhalb der Job-Definition beschrieben werden oder in einer separaten Datei. Beispiel 4.1, „Beispiel einer Job-Definition“ zeigt die eingebettete Variante aufgezeigt. Diese Indirektion wird auch gemacht, weil potenziell mehrere Job-Definitionen dieselbe Pipeline benützen können.

4.2.5. Vordefinierte Variablen

In den Jobs sind bestimmte Variablen von Anfang an gesetzt. Diese können nach Belieben verwendet werden, sollten aber wegen möglicher Nebeneffekte nicht überschrieben werden.

Vordefinierte Variablen

Variable	Beschreibung
{http://www.jeremias-maerki.ch/ns/wod/job}job-id	(String) Enthält die ID des aktuellen Jobs.
{http://www.jeremias-maerki.ch/ns/wod/job}job-uri	(URI) Enthält die URI des aktuellen Jobs.
{http://www.jeremias-maerki.ch/ns/wod/job}document	(URI) Enthält die URI des Hauptdokumentes des aktuellen Jobs.

4.3. Verarbeitungs-Pipelines

WOD stellt ein Pipeline-Subsystem zur Verfügung, mit dem Verarbeitungsketten auf einfache Weise zusammengestellt werden können. Eine Pipeline besteht aus einzelnen Pipes, welche jede eine bestimmte Teilaufgabe lösen. Beispiele für Pipes sind eine XSLT Transformation oder der Versand eines E-Mails.

Wo möglich, werden die einzelnen Pipes über die effizienteste Art miteinander verhängt. Beispielsweise kann eine Serialisierung einer XSL-FO Datei nach einer XSLT Transformation und das nachfolgende Parsen der Datei für die Formatierung gespart werden. Stattdessen wird der SAX Output der XSLT Transformation direkt als Input für die XSL-FO Formatierung verwendet. Ähnlich können auch externe Programme, die Eingabe und Ausgabe über Stdin und Stdout unterstützen, effizient eingebunden werden, ohne dass jeweils temporäre Dateien geschrieben werden müssen.

Der XML Namespace für Pipeline-Definitionen ist: `http://www.jeremias-maerki.ch/ns/wod/pipeline`

Nachfolgend werden sämtliche im Standard zur Verfügung stehende Pipes beschrieben. Je nach Installationsumfang kann allerdings nur ein Subset aller Pipes zur Verfügung stehen.

4.3.1. Pipes für Byte-Ströme

4.3.1.1. stream

Öffnet einen Byte-Strom basierend auf einer URI. Falls das Datenformat der geladenen Datei nicht automatisch identifiziert werden kann, kann es explizit angegeben werden.

```
stream (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
source	(Expr) URI des zu ladenden Dokuments.
format	(Optional, Expr) Format-URI oder MIME Typ des geladenen Formats. Standardwert: "application/octet-stream"

Verkettung

E/A	Mögliche Typen
Quelle	Start, Signal
Ziel	Byte-Strom

Beispiel 4.2. Beispiel für "stream"

```
<stream source="{job:representation(job:main-document())}"
    format="application/pdf"/>
```

Dieses Beispiel lädt die Haupt-Repräsentation des Haupt-Dokuments eines Jobs. Dass der Byte-Strom ein PDF ist wird mit dem `format` Attribut explizit angegeben.

4.3.1.2. stream-adapter

Technischer Adapter um einen ByteStreamProducer mit einem InputStreamConsumer zu verbinden.

```
stream-adapter (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
(keine)	

Verkettung

E/A	Mögliche Typen
Quelle	Byte-Strom (ByteStreamProducer)
Ziel	Byte-Strom (InputStreamConsumer)

Beispiel 4.3. Beispiel für "stream"

```
<apache-fop-formatter output-format="application/postscript"/>
<stream-adapter/>
<apply-ppd model-name="HP Universal Printing PS"/>
```

In diesem Beispiel wird der Stream Adapter dazu verwendet um Apache FOP mit dem PPD Prozessor zu verbinden. Auf Java-Ebene implementiert die FOP Pipe nur einen ByteStreamProducer, erwartet also einen ByteStreamConsumer als nachfolgende Pipe. Allerding bietet "apply-ppd" zur Zeit nur das InputStreamConsumer interface, weshalb die beiden Pipes nicht direkt miteinander verbunden werden können. Die Pipe "stream-adapter" überbrückt dies.

Ziel ist es, diese Pipe, wo nötig, automatisch einzuschiessen. Dies ist allerdings noch nicht implementiert.

4.3.1.3. stream-text

Codiert Text als Byte-Strom, wobei der Text eine Expression (Ausdruck) sein kann. Damit kann zum Beispiel Text für E-Mails mit personalisierten Elementen erstellt werden. Der Text wird in UTF-8 codiert (zur Zeit nicht parametrisierbar).

```
stream-text (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
Inhalt des Elements	(Expr) Zu codierender Text.
format	(Optional, Expr) Format-URI oder MIME Typ des produzierten Formats. Standardwert: "text/plain; charset=utf-8"

Verkettung

E/A	Mögliche Typen
Quelle	Start, Signal
Ziel	Byte-Strom

Beispiel 4.4. Beispiel für "stream"

```
<stream-text><![CDATA[{$anrede}

Angehängt finden Sie unsere Sonderangebote dieser Woche.

Freundliche Grüsse
Ihr Internet Shop XYZ]]></stream-text>
```

Dieses Beispiel lädt die Haupt-Repräsentation des Haupt-Dokuments eines Jobs. Dass der Byte-Strom ein PDF ist wird mit dem `format` Attribut explizit angegeben.

4.3.1.4. write-to-file

Schreibt einen Byte-Strom in eine Datei im lokalen Dateisystem

```
write-to-file (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
file	(String, Expr) Die Zieldatei.
create-missing-directories	(boolean, optional) Wird der Wert auf "true" gesetzt, werden allfällig fehlende Verzeichnisse erstellt. Ansonsten wird vorausgesetzt, dass das Zielverzeichnis existiert, und ein Fehler wird generiert, wenn es nicht existiert. Standardwert: false
safe-filenames	(boolean, optional) Wird der Wert auf "true" gesetzt, werden alle potenziell problematischen Zeichen aus dem Filenamen entfernt, so dass maximale POSIX-Kompatibilität gewährleistet ist. Es werden auch Umlaute entfernt. Standardwert: false
conflict-mode	(enum, optional) Wird der Wert auf "overwrite" gesetzt (oder weggelassen), wird eine existierende Zieldatei überschrieben. Wird der Wert auf "abort" gesetzt, schlägt die Pipe fehl, wenn die Zieldatei bereits existiert. Wird der Wert auf "adjust-filename" gesetzt, wird der Filename mit einem Index erweitert, so dass eine neue Datei geschrieben werden kann, ohne dass eine andere Datei überschrieben wird. Standardwert: overwrite

Verkettung

E/A	Mögliche Typen
Quelle	Byte-Strom
Ziel	Signal, Ende

Beispiel 4.5. Beispiel für "write-to-file"

```
<write-to-file file="D:/Temp/rechnung.pdf"/>
```

Dieses Beispiel schreibt einen Byte-Strom in die Datei `D:/Temp/rechnung.pdf`. Existiert die Datei bereits, wird sie in diesem Fall überschrieben.

4.3.2. Pipes für die XML Verarbeitung

4.3.2.1. parse-xml

Parst eine XML Datei identifiziert durch eine URI. Die Inhalte werden intern als SAX Event-Strom weitergegeben.

```
parse-xml (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
source	(Expr) URI des zu ladenden XML Dokuments.

Verkettung

E/A	Mögliche Typen
Quelle	Start, Signal
Ziel	SAX Events

Beispiel 4.6. Beispiel für "parse-xml"

```
<parse-xml source="{job:representation(job:main-document(), 'demo:invoice')}"/>
```

Dieses Beispiel lädt die 'demo:invoice' Repräsentation des Haupt-Dokuments eines Jobs.

4.3.2.2. serialize-xml

Serialisiert einen SAX Event-Strom und gibt dabei entweder einen Byte-Strom weiter oder schreibt eine Datei.

```
serialize-xml (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
target (optional)	(Expr) Die Ziel-URI (z.B. eine File URL).
output-format (optional)	(Expr) Das Ausgabeformat (MIME Typ) des generierten Formates. Hauptsächlich benützt bei Serialisierung zu einem Byte-Strom als Indikator für die nachfolgende Pipe.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Event-Strom
Ziel	Byte-Strom oder Signal/Ende

Beispiel 4.7. Beispiel für "serialize-xml"

```
<serialize-xml target="file:///D:/Temp/output.xml"/>
```

Dieses Beispiel serialisiert einen SAX Event-Strom in die Datei `D:/Temp/output.xml`.

4.3.2.3. validate-xml

Validiert eine XML Datei (üblicherweise mit XML Schema). Validiert wird ein SAX Event-Strom, der bei Bedarf unverändert weitergegeben wird.

Wenn keine Schemata angegeben werden, wird die XML Datei selber ausgewertet um allfällig verknüpfte Schemas zu laden.

```
validate-xml (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
schema-language (optional)	Die Schema-Sprache als Namespace URI. Der Standardwert ist 'http://www.w3.org/2001/XMLSchema' für XML Schema. Zur Verwendung von RelaxNG ist 'http://relaxng.org/ns/structure/1.0' anzugeben.

Kind-Elemente

Element	Beschreibung
schema*	Inhalt: URI (Expr). Eine oder mehrere URIs der zu ladenden Schema-Dateien.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Event-Strom
Ziel	SAX Event-Strom oder Signal/Ende

Beispiel 4.8. Beispiel für "validate-xml"

```
<validate-xml>
  <schema>http://localhost/schemas/invoice.xsd</schema>
  <schema>http://localhost/schemas/delivery-note.xsd</schema>
</validate-xml>
```

Dieses Beispiel validiert den eingehenden SAX Event-Strom gegen zwei XML Schemas.

4.3.2.4. xslt

Transformiert ein XML Dokument mittels XSLT.

```
xslt (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
stylesheet	(Expr) Die URI des XSLT Stylesheets.
processor (optional)	Der Name des zu verwendenden XSLT Prozessors, z.B. "xalan-j". Wird der Parameter nicht angegeben, wird der Standard-XSLT-Prozessor verwendet.

Kind-Elemente

Element	Beschreibung
param*	Inhalt: <leer>. Erlaubt die Angabe einer optionalen Liste von Parametern für das XSLT Stylesheet. Das Attribut 'name' gibt den Parameter-Namen an. Das Attribut 'value' gibt den zugehörigen Parameter-Wert an.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Event-Strom
Ziel	SAX Event-Strom

Beispiel 4.9. Beispiel für "xslt"

```
<xslt stylesheet="fo-replicator.xsl">
  <param name="repeats" value="10"/>
</xslt>
```

Dieses Beispiel transformiert einen SAX Event-Strom mit dem Stylesheet 'fo-replicator.xsl' und setzt dabei den XSLT Parameter 'repeats' auf '10'.

4.3.2.5. multi-xslt

Transformiert ein XML Dokument mittels XSLT. Allerdings wird hier nicht nur ein Stylesheet angewendet, sondern potenziell mehrere. Der eingehende XML Strom muss dafür `mxslt:transform` Elemente enthalten, welche das Stylesheet angeben und allenfalls XSLT Parameter für das Stylesheet.

Einsatz-Szenarien für diese Pipe ist zum Beispiel der Fall, dass es für mehrere Dokumententypen eigene Stylesheets gibt, die nicht ohne Probleme kombiniert werden können, damit sie gemeinsam arbeiten können. Man stelle sich in so einem Fall eine XML Datei vor, welches mehrere XML Dateien referenziert, die von unterschiedlichem Typ sind und demnach mit unterschiedlichen Stylesheets verarbeitet werden müssen. Aber alle sollen in einem Job verarbeitet werden. Zum Beispiel kann so ein grosser, heterogener XSL-FO Stream entstehen, der anschliessend formatiert wird.

```
multi-xslt (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
processor (optional)	Der Name des zu verwendenden XSLT Prozessors, z.B. "xalan-j". Wird der Parameter nicht angegeben, wird der Standard-XSLT-Prozessor verwendet.

Kind-Elemente

Element	Beschreibung
(keine)	

Verkettung

E/A	Mögliche Typen
Quelle	SAX Event-Strom
Ziel	SAX Event-Strom

Beispiel 4.10. Beispiel für "multi-xslt"

```
<parse-xml source="{job:representation(job:main-document(),
    'http://www.jeremias-maerki.ch/ns/demo/data')}"/>
<xslt stylesheet="multi-xslt-prepare.xsl">
<multi-xslt/>
<serialize-xml/>
```

Dieses Beispiel nimmt ein XML File aus Ausgang, transformiert es über XSLT, so dass es `<mxslt:transform>` Elemente enthält. Anschliessend übernimmt die `multi-xslt` Pipe die eigentliche Transformation über mehrere Stylesheets. Dessen Resultat wird serialisiert.

Nachfolgend wird ein vereinfachtes Beispiel gezeigt, welches die Verwendung des `<mxslt:transform>` Elements zeigt. Zunächst das mit den Transformierungs-Elementen angereicherte XML:

Beispiel 4.11. Beispiel für "multi-xslt": Eingabe-Strom

```
<?xml version="1.0" encoding="UTF-8"?>
<multi-root xmlns:mxslt="http://www.jeremias-maerki.ch/ns/multi-xslt">
  <title>Test Data 1</title>
  <mxslt:transform src="multi-xslt-invoice.xsl" param1="de">
    <invoice nr="123"/>
  </mxslt:transform>
  <mxslt:transform src="multi-xslt-delivery-note.xsl" param1="en">
    <delivery-note nr="124"/>
  </mxslt:transform>
</multi-root>
```

Wir haben hier also zwei Sub-Dokumente, eine Rechnung und einen Lieferschein, die jeweils mit unterschiedlichen Stylesheets verarbeitet werden sollen. Alle Attribute auf dem "transform" Element (ausser dem "src" Attribut) werden als XSLT Parameter an das Stylesheet übergeben. Es gilt zu beachten, dass die "transform" Elemente nicht geschachtelt werden dürfen. Wird dies dennoch getan, resultiert dies in einem Fehler.

4.3.2.6. build-dom

Erstellt eine DOM Struktur aus einem SAX Event-Strom. Das Resultat wird entweder direkt an eine Pipe weitergegeben, die DOMs verarbeiten kann, oder es wird als Kontextvariable gesetzt. Wenn eine Kontextvariable gesetzt wird, kann der SAX Event-Strom an die nächste Pipe weitergeleitet werden, oder aber es kann ein Signal an die nächste Pipe gesendet werden.

Das DOM Dokument kann beispielsweise später mittels der Pipe **use-dom** wiederverwendet werden.

```
build-dom (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
target-variable (optional)	(Expr) Name der Kontext-Variable, in die der DOM gespeichert werden soll.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Events
Ziel	SAX Events, Signal, DOM Consumer

Beispiel 4.12. Beispiel für "build-dom"

```
<build-dom target-property="jobticket"/>
```

Dieses Beispiel baut aus dem eingehenden SAX Event-Strom ein W3C DOM Objekt und setzt es als die Kontext-Variable 'jobticket'. Damit kann später auf einzelne Werte im XML über XPath zugegriffen werden, z.B. '{$jobticket/job:ticket/@id}'.

4.3.2.7. use-dom

Generiert einen SAX Event-Strom von einem W3C DOM Objekt. Diese Pipe ist das Gegenstück zu **build-dom**. Es bekommt das DOM Objekt entweder von der Vorgänger-Pipe oder via Expression (z.B. Kontext-Variable).

```
use-dom (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
source (optional)	(Expr) Die Quelle des DOM Objekts, üblicherweise eine Kontext-Variable (z.B. '{$jobticket}).

Verkettung

E/A	Mögliche Typen
Quelle	DOM Producer, Signal
Ziel	SAX Events

Beispiel 4.13. Beispiel für "use-dom"

```
<use-dom source="{$jobticket}"/>
```

Dieses Beispiel nimmt das DOM Objekt aus der Kontext-Variable 'jobticket' und generiert davon die nötigen SAX Events, die es an die nachfolgende Pipe weitergibt..

4.3.3. Pipes für das Aufsplitten von Dokumenten

4.3.3.1. split-xml

Splittet einen XML Strom in mehrere Dokumente auf. Damit der Splitter funktioniert, muss der eingehende XML Strom in einem bestimmten Format vorliegen, welches unten beschrieben wird. Vor dem Split wird also üblicherweise ein einfaches XSLT ausgeführt um den Split vorzubereiten.

```
split-xml
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
target-pipeline	(String) Name der Ziel-Pipeline, welche für jedes abgesplittete Dokument aufgerufen wird.
group-info	(String, optional) Mögliche Werte: "none", "pi", "root". Standartwert: "none". "pi" und "root" schreiben Informationen über die Split-Gruppe ins Ziel-Dokument. "start-index" enthält dabei den Index des ersten Dokuments über das ganze Ausgangsdokument gesehen. "group-index" enthält die 0-basierte Nummer der Gruppe. "pi" produziert am Anfang des Dokuments eine Processing Instruction namens "split-group-info". "root" schreibt diese Informationen stattdessen als Attribute in das Root Element des Dokuments.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Events (SAXProducer)
Ziel	keines (Finisher)

Beispiel 4.14. Beispiel für "split-xml"

```
<pipeline xmlns:jm="http://www.jeremias-maerki.ch/ns/xpath">
  <name>split-addresses</name>
  <variable name="cfg"
      value="{jm:get-configuration('com.acme.address-split')}"/>
  <variable name="package-size"
      value="{jm:value($cfg/split.package-size, 2000)}"/>
  <assembly>
    <parse-xml source="{$addresses-uri}"/>
    <xslt stylesheet="../xslt/addresses-split-prepare.xsl">
      <param name="package-size" value="{$package-size}"/>
    </xslt>
    <split-xml target-pipeline="split-addresses-store" group-info="root"/>
  </assembly>
</pipeline>

<pipeline xmlns:jm="http://www.jeremias-maerki.ch/ns/xpath">
  <name>split-addresses-store</name>
  <variable name="expiration"
      value="{jm:value($cfg/split.addresses.expiration, 'P7D')}"/>
  <assembly>
    <serialize-xml/>
    <store-stream expiration="{$expiration}"
       output-format="http://acme.com/addresses"
       identifier="{wod:identifier($source-uri)}-{wod:format-number($index, '000')}"/>
    <add-representation document="{$job:document}"/>
  </assembly>
</pipeline>
```

Dieses Beispiel splittet einen grösseren Strom von Adressen in Gruppen von z.B. 2000 Adressen auf. Die Adressen werden geparst, über XSLT der Split vorbereitete (siehe Stylesheet unten) und dann gesplittet. Für jede Split-Gruppe wird je einmal die Pipeline "split-addresses-store" aufgerufen, welche den SAX Strom serialisiert, ins Content Repository schreibt und als zusätzliche Repräsentation am Hauptdokument anhängt.

Die Split-Vorbereitung geschieht, wie bereits erwähnt, üblicherweise über ein einfaches XSLT Stylesheet. Das Root Element des Split-Formates ist "split:split" (Namespace: xmlns:split="http://www.jeremias-maerki.ch/ns/wod/xml/split"). Als erstes Kind wird das Element "split:group" erzeugt, welches die Struktur einer Split-Gruppe definiert. Im Beispiel unten wird am Format der Adressen nichts geändert. Das Element "split:insert-point" definiert dann, wo der pro Gruppe extrahiert Inhalt in die Gruppen-Vorlage eingebettet wird. Die einzelnen Gruppen werden über das Element "split:document" identifiziert. Der Inhalt eines "split:document" wird also anstelle eines "split:insert-point" im Ausgabestrom eingefügt.

Hier nun ein Beispiel des Eingabeformates:

Beispiel 4.15. XML Split: Address-Format

```
<addresses xmlns="http://acme.com/addresses">

  <recipient>
    <company>Muster AG</company>
    <given-name>Max</given-name>
    <family-name>Muster</family-name>
    <street>Musterstrasse</street>
    <zip>0000</zip>
    <place>Musterhausen</place>
  </recipient>

  <recipient>
    <given-name>Maria</given-name>
    <family-name>Bernasconi</family-name>
    <street>Beispielrain 2</street>
    <zip>1234</zip>
    <place>Entenhausen</place>
  </recipient>

  <recipient>
    <given-name>Clark</given-name>
    <family-name>Kent</family-name>
    <street>Farm 777</street>
    <zip>11111</zip>
    <place>Smallville</place>
    <country>USA</country>
  </recipient>

</addresses>
```

Hier nun das zugehörige XSLT Stylesheet für die Split-Vorbereitung:

Beispiel 4.16. XML Split-Vorbereitung

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:split="http://www.jeremias-maerki.ch/ns/wod/xml/split"
  xmlns:a="http://acme.com/addresses"
  xmlns="http://acme.com/addresses">

  <xsl:param name="package-size" select="2000"/>

  <xsl:template match="a:addresses">
    <split:split>
      <split:group max-documents="{$package-size}">
        <addresses>
          <xsl:copy-of select="@*"/>
```

```
          <split:insert-point/>
        </addresses>
      </split:group>
      <xsl:apply-templates/>
    </split:split>
  </xsl:template>

  <xsl:template match="a:recipient">
    <split:document>
      <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
      </xsl:copy>
    </split:document>
  </xsl:template>

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Das erzeugte Split-Dokument würde nach der XSLT Transformation und vor dem eigentlichen Split so aussehen:

Beispiel 4.17. Resultat der XML Split-Vorbereitung

```
<split:split
    xmlns:split="http://www.jeremias-maerki.ch/ns/wod/xml/split"
    xmlns="http://acme.com/addresses">
  <split:group max-documents="2000">
    <addresses>
      <split:insert-point/>
    </addresses>
  </split:group>

  <split:document>
    <recipient>
      <company>Muster AG</company>
      <given-name>Max</given-name>
      <family-name>Muster</family-name>
      <street>Musterstrasse</street>
      <zip>0000</zip>
      <place>Musterhausen</place>
    </recipient>
  </split:document>

  <split:document>
    <recipient>
      <given-name>Maria</given-name>
      <family-name>Bernasconi</family-name>
      <street>Beispielrain 2</street>
      <zip>1234</zip>
      <place>Entenhausen</place>
    </recipient>
  </split:document>

  <split:document>
    <recipient>
      <given-name>Clark</given-name>
      <family-name>Kent</family-name>
      <street>Farm 777</street>
      <zip>11111</zip>
      <place>Smallville</place>
      <country>USA</country>
    </recipient>
  </split:document>
```

```
</split:split>
```

4.3.4. Pipes für die Verarbeitung von Dokumentstruktur-Metadaten

4.3.4.1. build-document-structure

Baut Dokumentstruktur-Metadaten basierend auf Regel-Sets auf. Und zwar operiert die Pipe auf dem "Document Content" XML Format mit dem Namespace `http://www.jeremias-maerki.ch/ns/document/content`. Dieses Format wird z.B. von ??? generiert. Dies ist insbesondere nützlich, wenn PDF Dateien, welche anderswo formatiert/generiert wurden, mit Mediensteuerung versehen werden sollen.

```
build-document-structure
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
target-variable	(QName/String) Der Name der Variable, in der die Dokumentenstruktur gespeichert werden soll.

Kind-Elemente

Element	Beschreibung
page-position*	Definiert eine Regel, die in Abhängigkeit der Position der Seite angewendet wird. Das Attribut "position" kann die Werte "any", "first", "rest", "last" und "only" enthalten. Das Attribut "odd-or-even" kann die Werte "any", "odd" oder "even" enthalten. Die Semantik der beiden Attribute ist der von XSL-FO angelehnt.
xpath*	Definiert eine Regel, die über einen XPath Ausdruck prüft, ob diese anwendbar ist. Das Attribut "expr" enthält den XPath Ausdruckkann die Werte "any", "first", "rest", "last" und "only" enthalten. Das Attribut "odd-or-even" kann die Werte "any", "odd" oder "even" enthalten. Die Semantik der beiden Attribute ist der von XSL-FO angelehnt. Bei der Evaluierung des Ausdrucks ist jeweils der Seiten-Inhalt (`{http://www.jeremias-maerki.ch/ns/document/content}content`) das Kontext-Element (".").
page-position/set* xpath/set*	Setzt ein Property auf einem Struktur-Element. Das Attribut "target" kann die Werte "document", "recipient", "part" oder "page" enthalten. Das Attribut "name" definiert den Namen des Property, welches zu setzen ist. Das Attribut "value" enthält einen Ausdruck für den Wert des Property. Statt dem "name"/"value" Paar kann auch ein beliebiges Kind-Element erzeugt werden, wobei der qualifizierte Name des Kind-Elementes als Property-Namen verwendet wird.
page-position/begin-part xpath/begin-part	Zeigt an, dass mit der aktuellen Seite ein neues Teil-Dokument beginnt.
page-position/end-part xpath/end-part	Zeigt an, dass mit der aktuellen Seite das aktuelle Teil-Dokument endet und mit der nächsten Seite ein neues beginnt.
page-position/begin-recipient xpath/begin-recipient	Zeigt an, dass mit der aktuellen Seite ein neuer Empfänger beginnt.
page-position/end-recipient xpath/end-recipient	Zeigt an, dass mit der aktuellen Seite das aktuelle Empfänger endet und mit der nächsten Seite ein neuer beginnt.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Events (XML)

E/A	Mögliche Typen
Ziel	Signal oder DocumentStructureConsumer

Beispiel 4.18. Beispiel für "build-document-structure"

```
<?xml version="1.0" encoding="UTF-8"?>
<pipeline xmlns="http://www.jeremias-maerki.ch/ns/wod/pipeline">
  <name>test-structure1</name>
  <description>Test Pipeline: Test document structure build-up</description>
  <version>1.0</version>
  <assembly>
    <parse-xml source="{$source}"/>
    <build-document-structure target-property="structure"
            xmlns:regexp="http://exslt.org/regular-expressions">
      <xpath expr="{true()}">
        <set target="page" name="media-name" value="plain"/>
        <set target="document">
          <features xmlns="http://www.jeremias-maerki.ch/ns/postscript">
            <feature name="*Resolution" value="600dpi"/>
          </features>
        </set>
        <set target="page" name="is-first-page" value="{$is-first-page}"/>
        <set target="page" name="is-last-page" value="{$is-last-page}"/>
        <set target="page" name="is-only-page" value="{$is-only-page}"/>
        <set target="page" name="is-rest-page" value="{$is-rest-page}"/>
        <set target="page" name="is-odd-page" value="{$is-odd-page}"/>
        <set target="page" name="is-even-page" value="{$is-even-page}"/>
        <set target="page" name="page-number" value="{$page-number}"/>
        <set target="page" name="is-second-page" value="{$page-number = 2}"/>
      </xpath>
      <xpath expr="{regexp:test(string(.), '(\d{2}(\d{10})?\d\>)(\d{27}\+)\x20(\d{9}\>)')}">
        <begin-part/>
        <set target="page" name="media-name" value="esr"/>
      </xpath>
      <page-position position="first">
        <set target="page" name="pos" value="first"/>
      </page-position>
      <page-position position="last">
        <set target="page" name="pos" value="last"/>
      </page-position>
      <page-position position="only">
        <set target="page" name="pos" value="only"/>
      </page-position>
      <page-position position="rest">
        <set target="page" name="pos" value="rest"/>
      </page-position>
      <page-position position="any">
        <set target="page" name="any" value="true"/>
      </page-position>
      <page-position odd-or-even="odd">
        <set target="page" name="odd-or-even" value="odd"/>
      </page-position>
      <page-position odd-or-even="even">
        <set target="page" name="odd-or-even" value="even"/>
      </page-position>
    </build-document-structure>
    <document-structure-to-xml/>
    <write-to-file file="{$target-file}"/>
  </assembly>
</pipeline>
```

Dieses Beispiel, ein Unit Test, zeigt verschiedenste Anwendungsmöglichkeiten der oben beschriebenen Funktionselemente. Um ein Beispiel herauszupicken: Ein "xpath" Test prüft im Seiteninhalt, ob eine ESR-Codierzeile (Teil eines Schweizerischen Einzahlungsscheins) mittels Regular Expression gefunden werden kann. Falls dies zutrifft, wird für diese Seite der Medientyp "esr" gesetzt, und es wird mit dieser Seite ein neuer Dokumententeil begonnen.

Folgende spezielle Parameter stehen für Ausdrücke zur Verfügung:

- $is-first-page: (boolean) "true", wenn die aktuelle Seite die erste Seite des Dokuments ist.
- $is-last-page: (boolean) "true", wenn die aktuelle Seite die letzte Seite des Dokuments ist.
- $is-only-page: (boolean) "true", wenn die aktuelle Seite die einzige Seite des Dokuments ist.
- $is-rest-page: (boolean) "true", wenn die aktuelle Seite weder erste, die letzte Seite, noch die einzige des Dokuments ist.
- $page-number: (integer) Die aktuelle Seitennummer (1-basiert).
- $page-count: (integer) Die Anzahl Seiten des Dokuments, falls bekannt. Dieser Wert ist nicht immer verfügbar.
- $page-content: (Element) Der Inhalt der aktuellen Seite im "Document Content" Format. XML Element: `{http://www.jeremias-maerki.ch/ns/document/content}content`
- $page-resources: (Element) Die Resourcen der aktuellen Seite im "Document Content" Format. XML Element: `{http://www.jeremias-maerki.ch/ns/document/content}resources`

4.3.4.2. document-structure-to-xml

Konvertiert eine Dokumentenstruktur nach XML in das Format mit dem Namespace `http://www.jeremias-maerki.ch/ns/document/structure`. Die Pipe wird primär für Tests verwendet.

```
document-structure-to-xml
      (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
structure	(String, Expr, optional) Der Name der Variable, welche die Dokumentenstruktur enthält. Fehlt dieses Attribut wird erwartet, dass die vorhergehende Pipe diese Dokumentenstruktur erzeugt.

Verkettung

E/A	Mögliche Typen
Quelle	Signal oder DocumentStructureProducer
Ziel	SAX Events (XML)

Beispiel 4.19. Beispiel für "document-structure-to-xml"

```
<document-structure-to-xml structure="structure"/>
<store-stream expiration="P30D"
    output-format="http://www.jeremias-maerki.ch/ns/document/structure"
    identifier="{$name}.struct"/>
<add-representation document="{$job:document}"/>
```

Hier wird die Dokumentenstruktur aus der Variable "structure" nach XML konvertiert und als neue Repräsentation dem aktuellen Job-Dokument angehängt.

Ein weiteres Beispiel ist unter Beispiel 4.18, „Beispiel für "build-document-structure"“ zu finden.

4.3.5. Pipes für DocGenNG

4.3.5.1. docgen

Ruft DocGenNG für die Verarbeitung von Textverarbeitungs-basierten Dokument-Templates. Mit DocGenNG können z.B. in Microsoft Word, OpenOffice, LibreOffice u.a. Vorlagen/Templates mittels einer normalen Textverarbeitung geschrieben werden. Mit dieser Pipe lässt sich diese Funktionalität in WOD integrieren.

Es gilt zu beachten, dass die unterstützten Ausgabeformate abhängig vom Format der Vorlage sind.

Bei ODT oder OOXML wird in jedenm Fall ein Byte-Strom erzeugt. Im Falle von Word ML 2003 können neben einem Byte-Strom auch SAX Events erzeugt werden.

```
docgen
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
template	(URI, Expr) URI bzw. Dateipfad der Vorlage (Template).
output-format	(Expr) URI bzw. MIME Typ des gewünschten Ausgabeformates, z.B. "application/vnd.oasis.opendocument.text" für ODT.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Events (XML)
Ziel	Byte-Strom (z.B. ODT) oder SAX Events (nur Word ML 2003)

Beispiel 4.20. Beispiel für "docgen"

```
<parse-xml source="{job:representation(job:main-document(),
    'http://www.jeremias-maerki.ch/ns/test/rechnung')}"/>
<docgen template="rechnung1.odt"
    output-format="application/vnd.oasis.opendocument.text"/>
<store-stream expiration="P1Y"
    output-format="application/vnd.oasis.opendocument.text"
    identifier="{wod:identifier($source-uri)}"/>
```

Dieses Beispiel holt sich Rechnungs-Rohdaten vom Job, welche durch DocGenNG geschleust werden um eine ODT Datei zu erzeugen.

4.3.6. Pipes für PostFinance E-Rechnungen

4.3.6.1. yellowbill-upload

Lädt eine E-Rechung hoch zu Yellowbill von PostFinance AG.

```
yellowbill-upload
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
xml-source	(URI, Expr) Die URI für die XML Rohdaten im Yellowbill Invoice 2.0 Format.
pdf-source	(URI, Expr) Die URI für das PDF (Rechnungsdetail), welches mit der E-Rechnung mitgeschickt werden soll.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Beispiel 4.21. Beispiel für "yellowbill-upload"

```
<yellowbill-upload xml-source="{$ebill}" pdf-source="{$ebill-pdf}"/>
```

Das Beispiel lädt eine E-Rechnung hoch zu PostFinance AG, wobei das XML in der Variable `$ebill` und das PDF in der Variable `$ebill-pdf` vorgängig gesetzt wurde.

4.3.6.2. yellowbill-process-protocol-download

Lädt ein Verarbeitungsprotokoll für E-Rechnungen von PostFinance AG herunter. Die Pipe produziert einen XML Strom mit den heruntergeladenen Protokolldaten.

```
yellowbill-process-protocol-download
      (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
biller-id	(String, Expr) Die Biller ID des Rechnungssteller-Kontos, für welches das Verarbeitungsprotokoll abgerufen werden soll.
create-date	(xs:date oder Enum, optional) Gibt an, für welchen Zeitraum das Verarbeitungsprotokoll abgerufen werden soll. Es kann entweder direkt ein bestimmter Tag (Datum im xs:date Format) angegeben, oder es kann "today" (für heute) und "yesterday" für gestern angegeben werden. Wird dieses Attribut weggelassen, werden alle verfügbaren Einträge verarbeitet.
archive-data	(boolean, optional) Wenn auf "true" gesetzt, werden auch schon abgeholte Daten erneut abgeholt. Standardmässig werden nur neue Daten abgeholt.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	SAX Events (XML)

Nebeneffekte

Effekt	Beschreibung
Property "yellowbill.transaction.id"	Enthält die Transaktions-ID der hochgeladenen Rechnung.

Beispiel 4.22. Beispiel für "yellowbill-process-protocol-download"

```
<yellowbill-process-protocol-download biller-id="1234" create-date="yesterday"/>
<write-to-file file="{$target-file}"/>
```

Das Beispiel lädt das Verarbeitungsprotokoll für den vergangenen Tag für den (fiktiven) Rechnungssteller "1234" herunter und schreibt den resultierenden XML Strom in eine Datei.

Die resultierende XML Datei kann etwa so aussehen:

```
<?xml version="1.0" encoding="utf-8"?>
<process-protocols biller-id="1234" archive="false">
  <file filename="file1.xml">
    <Envelope>
      <Header>
        <From>IPEC</From>
        <To>Acme Corp.</To>
        <UseCase>GetProcessProtocol</UseCase>
        <SessionID/>
        <Version>1.1</Version>
        <Status>0</Status>
      </Header>
      <Body>
        <BillerID>411010000003xxxxx</BillerID>
        <DeliveryDate date="20151109">
          <NumberBills>1</NumberBills>
          <OK_Signed>1</OK_Signed>
          <OK_Result>
            <Signed>1</Signed>
            <Bill>
              <TransactionID>962cb580-37ec-4394-b877-f05653c9967b</TransactionID>
            </Bill>
          </OK_Result>
          <NOK_Result>
            <NotSigned>0</NotSigned>
          </NOK_Result>
        </DeliveryDate>
      </Body>
    </Envelope>
  </file>
</process-protocols>
```

Die heruntergeladenen Protokolle (XML Element "Envelope") werden in eine XML Datei zusammengefasst und können so als Ganzes verarbeitet werden. Die Details zum Verarbeitungsprotokoll sind dem Handbuch zur PostFinance E-Rechnung zu entnehmen.

4.3.6.3. yellowbill-extract-pdf

Extrahiert die PDF Datei aus einem heruntergeladenen Signatur-Paket (durch PostFinance signierte E-Rechnung).

```
yellowbill-extract-pdf
      (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
validate	(boolean, optional) Ein Wert "false" in diesem Attribut schaltet die standardmässige Signaturprüfung während der Verarbeitung ab.

Verkettung

E/A	Mögliche Typen
Quelle	InputStream
Ziel	Byte Stream

Beispiel 4.23. Beispiel für "yellowbill-extract-pdf"

```
<stream source="{job:representation(job:main-document())}"
    format="application/signature+xml"/>
<write-to-file file="D:/yellowbill-{$biller.id}_{$transaction.id}.xml"/>

<stream source="{job:representation(job:main-document())}"
    format="application/signature+xml"/>
<yellowbill-extract-pdf/>
<write-to-file file="D:/yellowbill-{$biller.id}_{$transaction.id}.pdf"/>
```

Das "format" Attribut sollte hier angegeben werden, da die "stream" Pipe momentan das Format noch nicht automatisch ermitteln kann.

Das Beispiel schreibt das originale Signatur-Paket als XML-Datei auf das lokale File System und extrahiert zusätzlich das PDF aus dem Signatur-Paket und schreibt es ebenfalls als separate Datei auf das lokale File System.

4.3.7. Pipes für den Aufruf externer Programme

4.3.7.1. exec

Ruft ein externes Programm auf. Es ist dabei möglich auf Ein- und Ausgabeseite einen Byte-Strom einzuhängen, wenn dies das externe Programm unterstützt (stdin/stdout piping).

```
exec
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
executable	(String, Expr) Name (und optional Pfad) des externen Programms.
dir	(String, Expr, optional) Der Pfad von welchem aus das externe Programm gestartet wird.

Kind-Elemente

Element	Beschreibung
arg*	(String, Expr, 0..n) Kommandozeilenparameter.

Verkettung

E/A	Mögliche Typen
Quelle	Signal oder Byte-Strom
Ziel	Signal oder Byte-Strom

Beispiel 4.24. Beispiel für "exec"

```
<stream source="{job:representation(job:main-document(),
        'ContentType:application/postscript')}"
     format="ContentType:application/postscript"/>
<exec dir="D:/Temp"
    executable="C:\Program Files\GhostScript\gs9.05\bin\gswin64c.exe"
    output-format="application/pdf">
  <arg>-dSAFER</arg>
  <arg>-dBATCH</arg>
  <arg>-dNOPAUSE</arg>
  <arg>-sstdout=%stderr</arg>
  <arg>-sDEVICE=pdfwrite</arg>
  <arg>-sOutputFile=-</arg> <!-- output to stdout -->
  <arg>-</arg> <!-- input from stdin -->
</exec>
<store-stream expiration="PT2H" identifier="{wod:identifier($source-uri)}"/>
```

Dieses Beispiel ruft GhostScript auf um ein PostScript Dokument nach PDF umzuwandeln. Die Ausgabe wird anschliessend im Standard Content Repository abgelegt.

4.3.8. Pipes für die Erzeugung von OSGi Events

4.3.8.1. send-event

Sendet einen Event an den OSGi EventAdmin service. Diese Pipe entspricht einem Aufruf von EventAdmin.sendEvent(), der Event wird also synchron produziert und verarbeitet. Die Verarbeitung wird erst fortgesetzt, wenn der Event erfolgreich verarbeitet wurde.

```
send-event
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
topic	(String) Name des Event Topic, z.B. "com/kunde/InfoEvent".
msg	(String, Expr, optional) Eine Expression für die Nachricht des Events.
(property/)name	(String) Der Name des Properties.
(property/)value	(String, Expr) Eine Expression für den Inhalt des Properties.

Kind-Elemente

Element	Beschreibung
property*	(String, Expr, 0..n) Ein Property mit Name und Wert (siehe Attribute oben).

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Beispiel 4.25. Beispiel für "send-event"

```
<send-event topic="com/acme/Printed" msg="Document {$doc-name} was printed.">
  <property name="doc-type" value="invoice"/>
</send-event>
```

Dieses Beispiel sendet einen Event für den Topic "com/acme/Printed" mit einer Nachricht und einem Property.

4.3.8.2. post-event

Sendet einen Event an den OSGi EventAdmin service. Diese Pipe entspricht einem Aufruf von EventAdmin.postEvent(), der Event wird also produziert und dann asynchron verarbeitet. Die Verarbeitung wird sofort fortgesetzt. Die Verarbeitung des Events geschieht parallel und ohne Rückmeldung.

```
post-event
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
topic	(String) Name des Event Topic, z.B. "com/kunde/InfoEvent".
msg	(String, Expr, optional) Eine Expression für die Nachricht des Events.
(property/)name	(String) Der Name des Properties.
(property/)value	(String, Expr) Eine Expression für den Inhalt des Properties.

Kind-Elemente

Element	Beschreibung
property*	(String, Expr, 0..n) Ein Property mit Name und Wert (siehe Attribute oben).

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Beispiel 4.26. Beispiel für "send-event"

```
<post-event topic="com/acme/Printed" msg="Document {$doc-name} was printed.">
  <property name="doc-type" value="invoice"/>
</send-event>
```

Dieses Beispiel sendet einen Event für den Topic "com/acme/Printed" mit einer Nachricht und einem Property.

4.3.9. Pipes für die Ablaufsteuerung

4.3.9.1. variable

Setzt eine Variable im Pipeline Context.

```
variable
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
name	(String/QName) Name der Variable (gültiger XML "Name", z.B. "name" oder "my:var1")
value	(Expr) Die Expression für den Variablen-Wert.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Beispiel 4.27. Beispiel für "variable"

```
<variable name="foo" value="bar"/>
<variable xmlns:t="test:test" name="t:id" value="{$foo}"/>
```

In diesem Beispiel wird zuerst die Variable "foo" mit dem Wert "bar" besetzt. Anschliessend wird die Variable "test" im Namespace "test:test" mit dem Wert der Variable "foo" besetzt.

4.3.9.2. call-pipe

Ruft eine andere Pipeline auf. Dabei wird die untergeordnete Pipeline mit dem Haupt-Context ausgeführt, also sind z.B. alle Variablen auch in der aufgerufenen Pipeline verfügbar.

```
call-pipe
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
name	(String) Name der aufzurufenden Pipeline. Nur erforderlich, falls keine eingebettete Pipeline spezifiziert ist.
if	(Boolean-Expression) Optionale Expression, die zu einem Boolean auflösen sollte. Die Pipe wird dann nur aufgerufen, falls die Expression den Wert "true()" zurückliefert.

Kind-Elemente

Element	Beschreibung
	(Elemente für Pipes, 0..n) Eine Liste von Pipes, die zu einer Pipeline zusammengefasst werden.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Beispiel 4.28. Beispiel für "call-pipe"

```
<call-pipe pipeline="print"/>
```

Dieses Beispiel ruft eine Pipeline namens "print" auf.

Beispiel 4.29. Beispiel für "call-pipe" mit eingebetteter Pipeline

```
<call-pipe if="{$mode = 1}">
  <variable name="foo" value="bar"/>
</call-pipe>
```

Dieses Beispiel ruft eine eingebettete Pipeline auf, falls die Variable "mode" den Wert "1" hat.

4.3.9.3. switch

Erlaubt eine Verzweigung in andere Pipelines über verschiedene Konditionen. Die einzelnen Konditionen werden in der vorgegebenen Reihenfolge abgeprüft. Die erste Kondition, die zutrifft, wird gewählt und dessen Pipeline aufgerufen.

```
switch
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
(case/)target-pipeline	(String) Name der aufzurufenden Pipeline. Nur erforderlich, falls keine eingebettete Pipeline spezifiziert ist.
(case/)condition	(Boolean, Expr) Die Kondition, die abgeprüft wird.

Kind-Elemente

Element	Beschreibung
case+	(Leer) Attribute wie oben beschrieben. Die "case" Elemente können statt dem Attribut "target-pipeline" auch eine Pipeline einbetten.
otherwise?	(Leer) Das "otherwise" Element kann statt dem Attribut "target-pipeline" auch eine Pipeline einbetten.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Beispiel 4.30. Beispiel für "switch"

```
<switch>
  <case target-pipeline="mail-pdf" condition="{boolean($email)}"/>
  <case target-pipeline="fax-pdf" condition="{boolean($fax)}"/>
  <otherwise target-pipeline="print-pdf"/>
</switch>
```

Dieses Beispiel ruft die Pipeline "mail-pdf" auf, wenn es im Kontext eine Variable "email" hat. Ähnlich für "fax-pdf". "{true()}" wird schlussendlich für den Fall verwendet, wo es weder eine E-Mail Addresse noch eine Faxnummer gibt.

Beispiel 4.31. Beispiel für "switch"

```
<switch>
  <case target-pipeline="mail-pdf" condition="{boolean($email)}">
    <send-mail from="noreply@acme.com">
      <to>{$email}</to>
      [..]
    </send-mail>
  </case>
  <case target-pipeline="fax-pdf" condition="{boolean($fax)}">
    <send-mail from="noreply@acme.com">
      <to>{$fax}@acme-faxservice.com</to>
      [..]
    </send-mail>
  </case>
  <otherwise>
    <stream source="{job:representation(job:main-document())}"
        format="application/pdf"/>
    <raw-print hostname="NPI41860F"/>
  </otherwise>
</switch>
```

Dieses Beispiel ist praktisch dasselbe Beispiel wie das erste, aber hier sind die aufgerufenen Pipelines eingebettet.

4.3.9.4. fail

Bricht die Verarbeitung der Pipeline mit einer eigenen Fehlermeldung ab.

```
fail
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
message	(Expr) Die Fehlermeldung.
if	(Boolean, Expr) Optionale Expression, die zu einem Boolean auflösen sollte. Die Pipe schlägt nur dann fehl, falls die Expression den Wert "true()" zurückliefert.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	-

Beispiel 4.32. Beispiel für "fail"

```
<fail message="Drucker {$printer-name} nicht unterstützt!"/>
```

4.3.10. Pipes für den HTTP Anfragen

4.3.10.1. http

Erzeugt eine HTTP POST oder PUT Anfrage.

```
http
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
url	(URL, Expr) Die HTTP URL der Anfrage.
method	("POST" oder "PUT", optional) Die HTTP Methode. Wird die Methode nicht angegeben, wird "POST" verwendet.
streamed	(Boolean, optional) Der Standardwert hier ist "true" und bedeutet, dass der Payload in den Aufruf "gestreamed" wird. Es findet also keine Pufferung statt. "false" puffert den Payload bevor er gesendet wird.

Kind-Elemente

Element	Beschreibung
header*	(leer bzw. String) Hiermit können zusätzliche HTTP Header für die Anfrage gesetzt werden. Der Header Name wird im Attribut "name" gesetzt, dessen Wert im Attribut "value" (oder im Text des Element-Inhalts).

Verkettung

E/A	Mögliche Typen
Quelle	Byte-Strom (InputStreamProducer)
Ziel	Byte-Strom (ByteStreamConsumer) oder Signal

Beispiel 4.33. Beispiel für "http"

```
<stream-text format="ContentType:application/json">
      \{"document-uri":"{$job:generated-content}"\}</stream-text>
<http url="{$job:callback-base}rest/invoice/{$doc/inv:invoice/@id}/document"
        method="PUT" streamed="false">
    <header name="AuthToken" value="{$auth-token}"/>
</http>
```

Dieses Beispiel produziert ein JSON Paket, welches mit HTTP PUT an einen REST Server übertragen wird. Als zusätzlicher HTTP Header wird beispielhaft ein Authentication Token übergeben, welches im Job Kontext als Variable gespeichert ist.

Erwähnenswert ist hier die teilweise Benützung von Backslashes zum Escaping der geschwungenen Klammern, die sonst für Expressions verwendet werden. Do können geschwungene Klammern für das JSON Paket und die Expressions voneinander getrennt werden.

TODO: Document HTTP authentication topics.

4.3.11. Pipes für die Dokument-Ablage bzw. -Auslieferung

4.3.11.1. store-stream

Legt ein Dokument in einem Dokumenten-Repository (DMS oder Archiv) ab.

```
store-stream (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
identifier (optional)	(Expr) Identifier (Name) des Dokuments, wenn ein expliziter Name zugeordnet werden soll. Ist dieser Parameter abwesend, wird ein geeigneter, automatischer Wert ausgewählt, der abhängig vom Kontext ist.
output-format (optional)	(Expr) URI bzw. MIME Typ des Dokument-Formates, z.B. "application/pdf". Dieser Parameter wird nur gesetzt, wenn die vorherige Pipe nicht das richtige Dokumentenformat mitteilt.
expiration (optional)	(Expr, ISO 8601 Dauer) Aufbewahrungsdauer (z.B. "P10Y") für das Dokument, wenn es nicht unbeschränkt aufbewahrt werden soll. Damit dies funktioniert, muss das darunter liegende Dokumenten-Repository ein solches Ablaufdatum unterstützen.
store-before-forward (optional)	(Expr, Boolean) Gibt an, ob der Dokumentinhalt zuerst gespeichert wird, bevor er an nachfolgende Pipes weitergeleitet wird (z.B. für den Druck). Damit kann beispielsweise sichergestellt werden, dass ein Dokument vollständig und korrekt erstellt wurde, bevor Daten an einen Drucker gesendet werden und dieser anfängt zu drucken, während noch weitere Seiten erstellt werden. Ist dies kein Problem, kann das Feature für eine höhere Performance abgeschalten werden, wodurch die Daten gleichzeitig ins Archiv und an die nachfolgende Pipe gestreamt wird. Standard: "true"
content-target-variable (optional)	(Name) Definiert die Variable, in welche die Content URI des gespeicherten Dokuments abgelegt wird. Standard: job:generated-content.

Verkettung

E/A	Mögliche Typen
Quelle	Byte Strom
Ziel	Byte Strom, Signal

Nebeneffekte

Effekt	Beschreibung
Variable "job:generated-content"	Enthält die URI des gespeicherten Dokuments. Wichtig: ersetzt einen bestehenden Wert! Der Name der Variable kann mit dem "content-target-variable" Attribut oben überschrieben werden.
Variable "job:generated-representation"	Enthält ein Repräsentations-Objekt, welches auf das gespeicherten Dokument zeigt. Dies wird beispielsweise von der Abschnitt 4.3.13.3, „add-representation“ Pipe verwendet. Wichtig: ersetzt einen bestehenden Wert!

Beispiel 4.34. Beispiel für "store-stream"

```
<stream source="{job:representation(job:main-document(),
    'http://www.jeremias-maerki.ch/ns/demo/telco')}"/>
<parse-xml/>
<xslt stylesheet="rechnung2fo.xsl"/>
<apache-fop-formatter output-format="application/postscript"/>
<ps-media-selection/>
<apply-ppd/>
<store-stream expiration="P2D"
  identifier="{wod:identifier($source-uri)}"/>
<print/>
```

Dieses Beispiel formatiert eine XML Datei via XSL-FO nach PostScript, speichert den PostScript-Strom im Dokumenten-Repository und leitet es anschliessend an den gewählten Drucker weiter. Der PostScript-Strom wird dabei 2 Tage lang gespeichert und wird anschliessend zur Löschung freigegeben.

4.3.11.2. deliver

Liefert ein Dokument an einen bestimmten Ort aus. Der Auslieferung findet mit Hilfe eines Delivery Services aus, der über einen Namen identifiziert wird.

???TODO Delivery Services dokumentieren!

```
deliver (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
document	(Expr) URI eines Dokumente, welches ausgeliefert werden soll.
target	(Expr) Name des Delivery Service, mit dem das Dokument ausgeliefert werden soll.
fail-on-missing-service (optional)	(Boolean) Definiert, ob die Pipe fehlschlagen soll, wenn der durch den Parameter *target* identifizierte Delivery Service nicht gefunden wird. Standard: "true"
retry-for (optional)	(Expr, ISO 8601 Dauer) Gibt an, wie lange versucht werden soll, das Dokument auszuliefern. Wird der Parameter nicht angegeben, wird unbeschränkt lange wieder und wieder versucht. Dies wird beispielsweise verwendet, wenn z.B. ein SFTP Server nicht immer garantiert zur Verfügung steht.
minimum-wait-between-retries (optional)	(Expr, ISO 8601 Dauer) Gibt an, wie lange mindestens gewartet wird, bis ein neuer Ausliefer-Versuch gestartet wird. Standard: 30 Sekunden.
maximum-wait-between-retries (optional)	(Expr, ISO 8601 Dauer) Gibt an, wie lange höchstens gewartet wird, bis ein neuer Ausliefer-Versuch gestartet wird. Standard: 5 Minuten.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Nebeneffekte

Effekt	Beschreibung
Property "job:drop-off-point"	Enthält die URI des Abliefert-Ortes, falls es einen solchen gibt, z.B. eine SFTP URL. Wichtig: ersetzt einen bestehenden Wert!

Beispiel 4.35. Beispiel für "deliver"

```
<deliver document="{$pdf-uri}" target="print-shop" retry-for="PT2H"/>
```

Dieses Beispiel versucht während 2 Stunden, das angegebene Dokument mit dem "print-shop" Delivery Service auszuliefern. Hinter dem Delivery Service können ganz verschiedene Mechanismen stecken, zum Beispiel Ablage ins lokale Dateisystem oder Versand via SFTP.

4.3.12. Pipes für Apache FOP (XSL-FO Verarbeitung)

4.3.12.1. apache-fop-formatter

Formatiert ein XSL-FO Dokument und gibt entweder einen Byte-Strom mit der generierten Datei aus oder einen XML-Strom mit dem FOP-eigenen Intermediate Format.

```
apache-fop-formatter
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
output-format	(Expr) URI bzw. MIME Typ des Ausgabeformates für den Formatierer, z.B. "application/pdf".
produce-if	(Boolean, optional) "true" um einen XML-Strom im FOP-eigenen Intermediate Format zu generieren. Dabei benützt die Pipe den "format" Parameter um die richtige Font-Konfiguration zu benützen, damit später das entsprechende Endformat ohne Verschiebungen beim Text produziert werden kann. "false" (Standardwert) produziert das angewählte Ausgabeformat.
config-name	(Expr, optional) Wenn gesetzt, wird die FOP Konfiguration mit dem angegebenen Namen für die Formatierung verwendet. Ist der Parameter nicht gesetzt, wird irgendeine Konfiguration verwendet oder eine Standard-Konfiguration, wenn keine andere gefunden wird.
structure-target-variable	(optional) Der Name der Zielvariable für die Dokumentenstruktur, wenn sie bei der Formatierung mit produziert werden soll.
xml:base	(Expr, optional) Eine Basis-URL, gegen welche relative URIs während der Formatierung aufgelöst werden.

Kind-Elemente

Element	Beschreibung
suppressed-events*/event*	Hier mit können bestimmte, von Apache generierte Events unterdrückt werden. Details dazu unter Abschnitt 4.3.12.4, „Behandlung von Apache FOP Events“.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Events (XML)
Ziel	Byte-Strom bzw. SAX Strom (beim Intermediate Format)

Beispiel 4.36. Beispiel für "fop-formatter"

```
<parse-xml source="{job:representation(job:main-document(),
    'demo:rechnung')}"/>
<xslt stylesheet="rechnung2fo.xsl"/>
<apache-fop-formatter output-format="application/postscript"
    config-name="client-xy" structure-target-variable="structure/>
```

Dieses Beispiel zeigt die Produktion einer Rechnung auf Basis einer XML-Eingangsdatei nach PostScript. Zuerst wird das XML aus dem Job gelesen, mittels XSLT nach XSL-FO konvertiert und schliesslich von Apache FOP in einen PostScript-Strom konvertiert. Dieser kann nach Belieben gedruckt, gespeichert oder anderweitig weiterverarbeitet werden.

4.3.12.2. apache-fop-format-to-pageable

Formatiert ein XSL-FO Dokument und produziert ein "java.awt.print.Pageable" Objekt, welches es erlaubt, XSL-FO Dokumente direkt über Betriebssystem-Drucker auszugeben.

```
apache-fop-format-to-pageable
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
config-name	(Expr, optional) Wenn gesetzt, wird die FOP Konfiguration mit dem angegebenen Namen für die Formatierung verwendet. Ist der Parameter nicht gesetzt, wird irgendeine Konfiguration verwendet oder eine Standard-Konfiguration, wenn keine andere gefunden wird.

Parameter	Beschreibung
structure-target-variable	(optional) Der Name der Zielvariable für die Dokumentenstruktur, wenn sie bei der Formatierung mit produziert werden soll.

Kind-Elemente

Element	Beschreibung
suppressed-events*/event*	Hier mit können bestimmte, von Apache generierte Events unterdrückt werden. Details dazu unter Abschnitt 4.3.12.4, „Behandlung von Apache FOP Events“.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Events (XML)
Ziel	java.awt.print.Pageable (PageableConsumer)

Beispiel 4.37. Beispiel für "apache-fop-format-to-pageable"

```
<parse-xml source="{job:representation(job:main-document(),
    'demo:rechnung')}"/>
<xslt stylesheet="rechnung2fo.xsl"/>
<apache-fop-format-to-pageable/>
<jps-print printer-name="Brother HL-1250" job-name="WOD Print Job"/>
```

Dieses Beispiel zeigt die direkten Druck einer Rechnung auf Basis einer XML-Eingangsdatei auf einem im Betriebssystem konfigurierten Drucker. Zuerst wird das XML aus dem Job gelesen, mittels XSLT nach XSL-FO konvertiert, über Apache FOP formatiert und via Betriebssystem auf den Drucker "Brother HL-1250" gedruckt.

4.3.12.3. apache-fop-if-renderer

Formatiert ein Dokument im FOP Intermediate Format in das gewählte Ausgabeformat und gibt das produzierte Dokument als Byte-Strom weiter.

```
apache-fop-if-renderer
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
output-format	(Expr) URI bzw. MIME Typ des Ausgabeformates für den Formatierer, z.B. "application/pdf".
config-name	(Expr, optional) Wenn gesetzt, wird die FOP Konfiguration mit dem angegebenen Namen für die Formatierung verwendet. Ist der Parameter nicht gesetzt, wird irgendeine Konfiguration verwendet oder eine Standard-Konfiguration, wenn keine andere gefunden wird.
structure-target-variable	(optional) Der Name der Zielvariable für die Dokumentenstruktur, wenn sie bei der Formatierung mit produziert werden soll.

Kind-Elemente

Element	Beschreibung
suppressed-events*/event*	Hier mit können bestimmte, von Apache generierte Events unterdrückt werden. Details dazu unter Abschnitt 4.3.12.4, „Behandlung von Apache FOP Events“.

Verkettung

E/A	Mögliche Typen
Quelle	SAX Events (Apache FOP Intermediate Format, ""application/X-fop-intermediate-format")

E/A	Mögliche Typen
Ziel	Byte-Strom

Beispiel 4.38. Beispiel für "apache-fop-if-renderer"

```
<parse-xml source="{job:representation(job:main-document(),
    'application/X-fop-intermediate-format')}"/>
<apache-fop-if-renderer format="application/pdf"/>
<write-to-file file="D:/Temp/out.pdf/>
```

Dieses Beispiel zeigt die Generierung eines PDF Dokuments von einem FOP Intermediate Format Strom. Die generierte PDF Datei wird dabei auf die Festplatte geschrieben.

4.3.12.4. Behandlung von Apache FOP Events

Alle Apache FOP Pipes erlauben es, die Behandlung der generierten Events anzupassen. Apache FOP erzeugt manchmal Events, die nicht unbedingt im Log erwünscht sind, da sie ev. gar keine Probleme darstellen, auch wenn FOP eine entsprechende Meldung erzeugt. Ein Beispiel ist der Event "org.apache.fop.fo.flow.table.TableEventProducer.paddingNotApplicable", der generiert wird, wenn z.B. ein Padding Property auf `fo:table-column` gesetzt wird. Zwar hat das auf die Tabellenspalte direkt noch keinen Effekt, aber dies zu tun ist absolut legitim, wenn man auf den Tabellenzellen mit der Funktion `from-table-column()` arbeitet. Hier also ein Beispiel, wie man diesen Event unterdrücken kann:

Beispiel 4.39. Beispiel für unterdrückte FOP Events

```
<apache-fop-formatter output-format="application/pdf">
  <suppressed-events>
    <event>org.apache.fop.fo.flow.table.TableEventProducer.paddingNotApplicable</event>
  </suppressed-events>
</apache-fop-formatter>
```

Details zum Event Framework von Apache FOP kann in dessen Dokumentation[2] nachgelesen werden. Um die Namen der zu unterdrückenden Events herauszufinden, kann das Job Log eingesehen werden, wo die Event ID jeweils angezeigt wird.

4.3.13. Pipes für die Manipulation von Jobs

4.3.13.1. create-job

Erstellt einen neuen Job.

```
create-job
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
job-type	(URI, Expr) Die URI des Job Typs.
main-document	(URI, Expr) Die URI des Hauptdokuments für den Job.
expires-after	(XML Schema Duration, optional) Die Dauer, bis der Job ausläuft und gelöscht werden kann. Der Wert wird in der Syntax einer XML Schema Duration angegeben, also z.B. "P3D" (3 Tage) oder "PT48H" (48 Stunden).
priority	(Ganzzahl 0..65535, optional) Die Priorität des Jobs. Standardwert: 0.

Kind-Elemente

Element	Beschreibung
attributes?	(Name/Wert Paare) Jedes Kind hat einen Namen und einen Wert. Der Name ist im Elementnamen codiert (mit oder ohne Namespace), der Wert im Inhalt des Elements.

[2] http://xmlgraphics.apache.org/fop/trunk/events.html

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Nebeneffekte

Effekt	Beschreibung
Property "wod:job-uri"	Enthält die URI des neu erstellten Jobs. Wichtig: ersetzt den bestehenden Wert!
Property "wod:job-id"	Enthält die ID des neu erstellten Jobs. Wichtig: ersetzt den bestehenden Wert!

Beispiel 4.40. Beispiel für "create-job"

```
<pipeline>
  <name>telco-split-store</name>
  <assembly>
    <serialize-xml/>
    <store-stream expiration="PT2H"
        output-format="http://www.jeremias-maerki.ch/ns/demo/telco/batch/format"
        identifier="{wod:identifier($source-uri)}-{wod:format-number($index, '000000')}"/>
    <create-document>
      <attributes>
        <origin xmlns="">telco-split-store</origin>
      </attributes>
    </create-document>
    <create-job job-type="http://www.jeremias-maerki.ch/ns/demo/telco/batch/format"
        main-document="{$job:document}" priority="0" expire-after="P3D">
      <attributes>
        <origin xmlns="">telco-split-store</origin>
      </attributes>
    </create-job>
  </assembly>
</pipeline>
```

Dieses Beispiel definiert eine vollständige Hilfs-Pipeline, welche die Ausgabe einer XML-Split Operation verarbeitet. Der XML Strom wird ins Content Repository gespeichert. Für das gespeicherte Dokument wird ein logisches Dokument erstellt und dieses an einen neuen Job angehängt. Der Job hat Priorität "0" und läuft nach 3 Tagen ab. Zusätzlich wird ein Job Parameter "origin" gesetzt.

4.3.13.2. create-document

Erstellt einen neues Dokument.

```
create-document
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
(keine)	

Kind-Elemente

Element	Beschreibung
attributes?	(Name/Wert Paare) Jedes Kind hat einen Namen und einen Wert. Der Name ist im Elementnamen codiert (mit oder ohne Namespace), der Wert im Inhalt des Elements.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Vorraussetzungen

Effekt	Beschreibung
Property "wod:generated-representation"	Dieses Property muss gesetzt sein, bevor diese Pipe aufgerufen werden kann. Dies kann z.B. über die Pipe "store-stream" geschehen, also über das Speichern eines Dokuments im Content Repository

Nebeneffekte

Effekt	Beschreibung
Property "job:document"	Enthält die URI des neu erstellten Dokuments. Wichtig: ersetzt den bestehenden Wert!

Als Beispiel sei auf Beispiel 4.40, „Beispiel für "create-job"“ verwiesen.

Falls der aktuelle Job kein Hauptdokument hat, wird das neu erstellte Dokument als Hauptdokument auf dem Job gesetzt.

4.3.13.3. add-representation

Fügt eine neue Repräsentation zu einem Dokument hinzu.

```
add-representation
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
document	(URI, Expr) Die URI des Dokuments, an welches die Repräsentation angehängt werden soll. Standardwert: "{$job:document}"
source	(URI, Expr) Die URI der Repräsentation (Quelle für das Dokument). Dies kann auch eine URL sein. Dieser Parameter ist optional, falls die Variable "job:generated-representation" gesetzt ist, was z.B. nach dem Aufruf einer Abschnitt 4.3.11.1, „store-stream“ Pipe der Fall ist.
format	(URI bzw. MIME Typ) Der Dokumenttype als URI oder MIME Typ. Dieser Parameter wird nur benötigt, falls "source" verwendet wird.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Beispiel 4.41. Beispiel für "add-representation"

```
<store-stream expiration="P3D"
    output-format="application/pdf"
    identifier="{wod:identifier($source-uri)}.pdf"/>
<add-representation document="{$job:document}"/>
```

Dieses Beispiel hängt das im Content Repository gespeicherte PDF Dokument als neue Repräsentation an das aktuelle Hauptdokument an.

Beispiel 4.42. Beispiel für "add-representation"

```
<add-representation document="{$job:document}"
    source="http://localhost/docs/123.pdf"
    format="application/pdf"/>
```

Dieses Beispiel zeigt die Verwendung mit expliziter URL statt mit "store-stream".

4.3.13.4. update-job

Aktualisiert den aktuellen Job.

Zur Zeit wird der Job nach der Änderung nicht direkt in der Datenbank aktualisiert! Dies geschieht erst am Schluss des Jobs durch den Job Handler.

```
create-job
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
main-document	(URI, Expr, optional) Überschreibt das Hauptdokument des Jobs.
expires-after	(URI, Expr, optional) Überschreibt das Ablaufdatum des Jobs.

Kind-Elemente

Element	Beschreibung
attributes?	(Name/Wert Paare) Jedes Kind hat einen Namen und einen Wert. Der Name ist im Elementnamen codiert (mit oder ohne Namespace), der Wert im Inhalt des Elements.

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal

Beispiel 4.43. Beispiel für "update-job"

```
<update-job>
  <attributes>
    <amount xmlns="">{$amount}</amount>
  </attributes>
</update-job>
```

Dieses Beispiel aktualisiert den aktuellen Job und setzt das Attribut "amount".

4.3.14. Pipes für Lokalisierung

4.3.14.1. l10n

Diese Pipe ist ein XML Filter für die Lokalisierung (Localization, L10n). Sie ersetzt Elemente aus dem l10n Namespace (http://jeremias-maerki.ch/l10n) mit der lokalisierten Version. Hiermit können also zum Beispiel mehrsprachige Dokument mit einem einzigen Template umgesetzt werden.

Für Details zur Syntax und Verwendung des Localization Filters sei auf Kapitel 11, *Lokalisierung (Localization, L10n)* verwiesen.

```
l10n
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
base-name	(String) Basis-Name für sämtliche Übersetzungen.

Parameter	Beschreibung
locale	(String, optional) Standard-Sprache. Standardwert: "en" bzw. Framework Property "osgi.nl")

Verkettung

E/A	Mögliche Typen
Quelle	SAX Events (XML)
Ziel	SAX Events (XML)

Die Standardsprache hier greift nur, falls die Sprache nicht innerhalb des eingehenden XML Dokuments über das `xml:lang` Attribute überschrieben wird.

Die Übersetzungsdateien müssen in demselben Bundle wie die Pipeline-Definition liegen. Ansonsten werden die Dateien nicht gefunden.

Beispiel 4.44. Beispiel für "l10n"

```
<parse-xml source="{job:representation(job:main-document(),
    'http://www.jeremias-maerki.ch/ns/demo/worklog')}"/>
<xslt stylesheet="worklog2fo-with-l10n.xsl"/>
<l10n base-name="trans" locale="de"/>
<apache-fop-formatter output-format="application/pdf"/>
```

Dieses Beispiel parst eine XML Datei, transformiert sie nach XSL-FO mit einem XSLT Stylesheet, übersetzt anschliessend das generierte Dokument und formatiert es mittels Apache FOP.

4.3.15. Pipes für die Generierung von E-Mails

4.3.15.1. send-mail

Verschickt ein E-Mail. Es ist möglich sowohl einen Plain Text Teil, wie auch einen HTML Teil zu erstellen und beliebige Attachments (z.B. PDF Dateien) anzuhängen.

```
send-mail
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
from	(String, Expr) Absender E-Mail Adresse z.B. "noreply@acme.com"
reply-to (optional)	(String, Expr) E-Mail Adresse, an die eine Antwort geschickt werden soll, falls diese von der "From" Adresse abweicht.
service (optional)	(String, Expr) Der Name des E-Mail Versand Service, der verwendet werden soll. Falls dieser Parameter gesetzt wird, ist das `<smtp>` Kind-Element zur Konfiguration des Mail-Servers nicht notwendig.
retry-for (optional)	(Expr, ISO 8601 Dauer) Gibt an, wie lange versucht werden soll, das Dokument auszuliefern. Wird der Parameter nicht angegeben, wird während 5 Minuten wieder und wieder versucht. Dies wird beispielsweise verwendet, wenn z.B. ein SMTP Server nicht immer zuverlässig zur Verfügung steht. Gelingt der Versand innerhalb dieser Zeit nicht, schlägt der aktuelle Job fehl. Wird dieser Wert auf z.B. "PT0S" gestellt, wird nicht nach einem Fehler nicht erneut versucht zu versenden.
minimum-wait-between-retries (optional)	(Expr, ISO 8601 Dauer) Gibt an, wie lange mindestens gewartet wird, bis ein neuer Ausliefer-Versuch gestartet wird. Standard: 30 Sekunden.

Parameter	Beschreibung
maximum-wait-between-retries (optional)	(Expr, ISO 8601 Dauer) Gibt an, wie lange höchstens gewartet wird, bis ein neuer Ausliefer-Versuch gestartet wird. Standard: 5 Minuten.

Kind-Elemente

Element	Beschreibung
to+	(String, Expr, 1..n) Empfänger Adresse.
cc*	(String, Expr, 0..n) CC Empfänger Adresse.
bcc*	(String, Expr, 0..n) BCC Empfänger Adresse.
subject	(String, Expr) Betreff.
text	(String, Expr) Plain Text Teil des E-Mails.
html (Namespace: `http://www.w3.org/1999/xhtml`)	(XHTML, Expr) XHTML Teil des E-Mails.
attachment*	(String, Expr, 0..n) URI der anzuhängenden Datei. Das optionale Attribut "filename" ermöglicht es einen expliziten Dateinamen für das Attachment anzugeben. Das optionale Attribut "format" erlaubt es den MIME Typ für das Attachment explizit anzugeben.
smtp?	(Element) Dieses Element enthält Informationen über den SMTP Server. Das Attribut "hostname" enthält den Hostnamen oder die IP Adresse des SMTP Servers. Das Attribut "port" erlaubt es den Standard-Port (25) zu überschreiben. Das Attribut "username" enthält den Usernamen, das Attribut "password" das Passwort für die SMTP Authentifizierung, falls benötigt. Das optionale Attribut "protocol" erlaubt entweder "smtp" (Standardwert) oder "smtps". Das optionale Attribut "tls" kann auf "true" gesetzt werden um das SMTP Kommando "STARTTLS" zu aktivieren. Statt dieses Elements kann mit dem "service" Parameter (siehe oben) ein E-Mail Versand Service verwendet werden, der anderswo konfiguriert wird.

Parametrisierung des "smtp" Elements

Parameter	Beschreibung
hostname	(String, Expr) Der Hostname des SMTP Servers.
port	(Integer, Expr, optional) Der Port des SMTP Servers (Standardwert: 25)
username	(String, Expr, optional) Der Username für die Authentifizierung am SMTP Server.
password	(String, Expr, optional) Das Passwort für die Authentifizierung am SMTP Server.
tls	(Boolean, Expr, optional) Wird dieser Parameter auf "true" gesetzt, wird das SMTP Kommando "STARTTLS" aktiviert, um die Verbindung zu verschlüsseln. (Bemerkung: das ist nicht dasselbe wie SMTPS)
protocol	(Enumeration, Expr) "smtp" für das SMTP Protokoll, "smtps" für das verschlüsselte SMTPS Protokoll (Standardwert: smtp)

Kind-Elemente des "smtp" Elements

Parameter	Beschreibung
property*	(Element) Ein "javax.mail" spezifisches Property zur feineren Konfiguration des E-Mail Versands. Das Attribut "name" enthält den Namen des Properties. Das Attribut "value" ist eine Expression für den Wert des Properties. Gültige Property Namen (z.B. "mail.smtp.auth") können im Internet gefunden[3] werden.

[3] https://javamail.java.net/nonav/docs/api/com/sun/mail/smtp/package-summary.html

Verkettung

E/A	Mögliche Typen
Quelle	Signal
Ziel	Signal, Byte-Strom

Es ist empfohlen, die SMTP Parameter via Framework Properties (%WOD_HOME%/etc/framework.properties) zu setzen, damit keine Passwörter in den Job Definitionen auftauchen.

Beispiel 4.45. Beispiel für "send-mail"

```
<send-mail from="noreply@acme.com">
  <subject>Lieferschein zur Bestellung Nr. {$order-nr}</subject>
  <text>Ihre Bestellung Nr. {$order-nr} wurde versandt.
Angehängt finden Sie den Lieferschein.</text>
  <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
      <title>HTML Mail Body</title>
    </head>
    <body>
      <p>Ihre Bestellung Nr. {$order-nr} wurde versandt.</p>
      <p>Angehängt finden Sie den Lieferschein.</p>
      <p><img src="inline:customer:e-mail-footer.png"/></p>
    </body>
  </html>
  <to>{$email}</to>
  <attachment>{job:representation(job:main-document(),
      "application/pdf")}</attachment>
  <smtp hostname="{$smtp.host}"
      username="{$smtp.username}" password="{$smtp.password}">
    <property name="mail.smtp.connectiontimeout" value="10000"/>
  </smtp>
</send-mail>
```

Dieses Beispiel hängt das im Verlaufe des Jobs erstellte PDF an ein E-Mail an, welches sowohl Plain Text wie auch XHTML enthält an und verschickt es an eine bestimmte E-Mail Adresse. Praktisch überall können Ausdrücke verwendet werden. Das Beispiel zeigt auch die Verwendung von Framework Properties für SMTP Host, Username und Passwort. Ausserdem zeigt es, wie das Connection Timeout für die SMTP Verbindung auf 10 Sekunden eingestellt werden kann.

Das Beispiel zeigt ebenfalls, wie Bilder in die verschickten E-Mails eingebettet werden können (z.B. Firmenlogos). Dazu kann in `<img>` Tags die Bild-URI mit dem Präfix "inline:" versehen werden. Der Text nach diesem Präfix ist dann die URI, mit der das Bild abgerufen wird für die Einbettung in die E-Mail Nachricht. Wird der Präfix weggelassen, erfolgt keine Einbettung. Das bedeutet aber, dass nur HTTP(S) URLs verwendet werden sollen, die öffentlich über das Internet zugänglich sind.

Beispiel 4.46. Beispiel für "send-mail" (Variante mit Versand-Service)

```
<send-mail from="noreply@acme.com" service="acme">
  <subject>Lieferschein zur Bestellung Nr. {$order-nr}</subject>
  <text>Ihre Bestellung Nr. {$order-nr} wurde versandt.
Angehängt finden Sie den Lieferschein.</text>
  <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
      <title>HTML Mail Body</title>
    </head>
    <body>
      <p>Ihre Bestellung Nr. {$order-nr} wurde versandt.</p>
      <p>Angehängt finden Sie den Lieferschein.</p>
    </body>
  </html>
  <to>{$email}</to>
  <attachment>{job:representation(job:main-document(),
      "application/pdf")}</attachment>
</send-mail>
```

Dieses Beispiel zeigt die Benützung eine E-Mail Versand Service statt dem `<smtp>` Element.

Ist die nachfolgende Pipe ein Byte-Strom Consumer, ist das "smtp" Element nicht notwendig. In diesem Fall wird das erstellte E-Mail im RFC 822 Format (message/rfc822 bzw. EML Format) an die nachfolgende Pipe übertragen.

E-Mail Services können z.B. über folgende Konfigurationen bereitgestellt werden:

- Abschnitt 2.7.4, „JM :: E-Mail :: Sender :: SMTP“
- Abschnitt 2.7.5, „JM :: E-Mail :: Sender :: File System“

4.3.16. Pipes für die Versand von Fax-Nachrichten

4.3.16.1. send-fax

Verschickt eine Fax Nachricht.

```
send-fax
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
recipient	(String, Expr) Telefonnummer des Empfängers. Sowohl normale Telefonnummern (z.B. E.164) wie auch tel: oder fax: URIs werden unterstützt. Beispiele: "+41413709458" oder "tel:+41-41-370-94-58"
service	(String) Name des Fax Service, der zum Versand benützt werden soll.
require-receipt (optional)	(boolean) Wird dieser Wert auf "true" gesetzt, führt eine fehlende Empfangsbestätigung zum Abbruch der Verarbeitung mit einem Fehler. Zu beachten: nicht alle Fax-Versand Services unterstützen Empfangsbestätigungen (z.B. ein E-Mail-zu-Fax Gateway).
timeout (optional)	(Expr, ISO 8601 Dauer) Gibt an, wie lange versucht werden soll, das Dokument auszuliefern. Wird der Parameter nicht angegeben, wird bis zu 5 Minuten gewartet. Läuft diese Zeit ab, wird mit einem Fehler abgebrochen. Zu beachten: trotz Abbruch kann es sein, dass die Nachricht (oder Teile davon) in der Zwischenzeit dennoch beim Empfänger angekommen sind.

Verkettung

E/A	Mögliche Typen
Quelle	InputStream, Images
Ziel	Signal

Beispiel 4.47. Beispiel für "send-fax"

```
<pipeline>
  <name>fax-pdf</name>
  <assembly>
    <stream source="{job:representation(job:main-document())}"
        format="application/pdf"/>
    <send-fax recipient="{$fax}" service="ecall"/>
  </assembly>
</pipeline>
```

In diesem Beispiel geschieht der Fax-Versand über einen Fax-Service mit dem Namen "ecall". Dabei wird die PDF Repräsentation des Hauptdokuments des aktuellen Jobs versandt.

Fax Services können z.B. über folgende Konfigurationen bereitgestellt werden:

- Abschnitt 2.7.7, „JM :: Fax :: Client :: File-system“
- Abschnitt 2.7.6, „JM :: Fax :: Client :: eCall E-Mail to Fax“

4.3.17. Pipes für die PDF Verarbeitung mit Apache PDFBox

4.3.17.1. pdfbox-parse-pdf

Lädt ein PDF Dokument mit Apache PDFBox[4].

```
pdfbox-parse-pdf
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
source	(URI, Expr, optional) URI des zu ladenden PDF Dokuments, falls nicht direkt eine Byte-Strom verarbeitet wird.

Verkettung

E/A	Mögliche Typen
Quelle	Signal oder Byte-Strom
Ziel	PDFBox Dokument (PDDocumentConsumer)

Beispiel 4.48. Beispiel für "pdfbox-parse-pdf"

```
<pdfbox-parse-pdf source="{job:representation(job:main-document())}"/>
```

Lädt das Hauptdokument des Jobs als PDF mit PDFBox zur weiteren Verarbeitung.

4.3.17.2. pdfbox-extract-text

Extrahiert Text aus einem PDF Dokument mittels Apache PDFBox. Die gewünschten Texte werden mittels Regular Expressions identifiziert und anschliessend in Kontext Variablen geschrieben.

```
pdfbox-extract-text
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
pages	(String, optional) Definiert die Seiten, von denen der Text extrahiert werden soll. Beispiele: "1" oder "1,3,7-8". Wird dieser Parameter weggelassen, werden sämtliche Seiten des PDFs verarbeitet.

Kind-Elemente

Element	Beschreibung
regex+	(String, Regular Expression, 1..n) Regular Expression (Syntax nach Java's java.util.regex.Pattern) um Werte aus dem extrahierten Text in eine Kontextvariable zu schreiben. Das "target-variable" Attribut bestimmt die Zielvariable.

Verkettung

E/A	Mögliche Typen
Quelle	PDFBox Dokument (PDDocumentProducer)
Ziel	Signal oder PDFBox Dokument (PDDocumentConsumer)

[4] http://pdfbox.apache.org

Beispiel 4.49. Beispiel für "pdfbox-extract-text"

```
<pdfbox-parse-pdf source="{job:representation(job:main-document())}"/>
<pdfbox-extract-text pages="1">
  <regex target-variable="email">(?&lt;=RecipientEmail\:).*</regex>
  <regex target-variable="fax">(?&lt;=SendFax\:).*</regex>
  <regex target-variable="username">(?&lt;=Username\:).*</regex>
</pdfbox-extract-text>
<update-job>
  <attributes>
    <email>{$email}</email>
    <fax>{$fax}</fax>
    <username>{$username}</username>
  </attributes>
</update-job>
```

Lädt ein PDF Dokument und versucht, eine E-Mail Adresse, eine Faxnummer und einen Usernamen aus dem PDF zu extrahieren und als Job Attribute zu speichern. Beispielsweise wird die E-Mail Adresse gemäss der Regular Expression nach folgendem Muster erwartet: "RecipientEmail:info@acme.com". Die Regular Expression gibt hierbei lediglich die E-Mail Adresse ohne den "RecipientEmail:" Marker aus. Dieser Text könnte zum Beispiel weiss auf weiss als Metadaten ins PDF geschrieben worden sein.

4.3.17.3. pdfbox-to-bitmap

Produziert Bitmaps von einer Serie von Seiten eines PDFs. Es können verschiedene Ausgabeformate wie JPEG, PNG oder TIFF angegeben werden. Im Falle von TIFFs können mehrere Seiten als eine Datei erstellt werden. Bei der Konversion einer einzelnen Seite können beispielsweise JPEGs oder PNGs direkt erzeugt werden. Werden Einzelbild-Formate wie JPEG oder PNG gewählt, aber mehrere Seiten ausgewählt, wird pro Seite eine Datei erzeugt und sämtliche Bitmaps in einen ZIP Container verpackt.

Ohne andere Parametrisierung produziert diese Pipe ein Bi-Level TIFF mit CCITT T.6 Kompression und einer Auflösung von 300 dpi.

```
pdfbox-to-bitmap
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
pages	(String, optional) Definiert die Seiten, von denen der Text extrahiert werden soll. Beispiele: "1" oder "1,3,7-8". Wird dieser Parameter weggelassen, werden sämtliche Seiten des PDFs verarbeitet.
image-format	(MIME Type, optional) Das Bitmap Ausgabeformat, z.B. "image/tiff", "image/jpeg" oder "image/png". Standardwert: "image/tiff"
compression	(String, optional) Der Kompressionstyp für das Bild. Mögliche Werte (nicht abschliessend): "CCITT T.4", "CCITT T.6", "PackBits", "LZW", "JPEG", "ZLib", "None". Standardwerte bei Bi-Level (1bit) Bildern is "CCITT T.6", anderenfalls "PackBits". Nicht alle Ausgabeformat unterstützen alle Kompressionstypen!
bits-per-pixel	(Ganzzahl, optional) Anzahl Bits pro Pixel, z.B. 1, 8 oder 24. Standardwert: 1 für Bi-Level.
resolution	(Ganzzahl, optional) Auflösung der Bilder in Pixel pro Zoll (dpi). Standardwert: 300.

Verkettung

E/A	Mögliche Typen
Quelle	PDFBox Dokument (PDDocumentProducer)
Ziel	Byte-Strom

Beispiel 4.50. Beispiel für "pdfbox-to-bitmap"

```
<pdfbox-parse-pdf source="{job:representation(job:main-document())}"/>
<pdfbox-to-bitmap pages="1" image-format="image/png" bits-per-pixel="24"/>
<write-to-file file="D:/Temp/rechnung.png"/>
```

Lädt ein PDF Dokument, produziert ein farbiges PNG der ersten Seite und schreibt dieses in eine Datei.

4.3.17.4. pdfbox-to-pageable

Produziert ein `java.awt.print.Pageable` Objekt von einer Serie von Seiten eines PDF Dokuments mit Apache PDFBox[5].

```
pdfbox-to-pageable
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
pages	(String, optional) Definiert die Seiten, von denen der Text extrahiert werden soll. Beispiele: "1" oder "1,3,7-8". Wird dieser Parameter weggelassen, werden sämtliche Seiten des PDFs verarbeitet.

Verkettung

E/A	Mögliche Typen
Quelle	PDFBox Dokument (PDDocumentProducer)
Ziel	java.awt.print.Pageable

Beispiel 4.51. Beispiel für "pdfbox-to-pageable"

```
<pdfbox-parse-pdf source="{job:representation(job:main-document())}"/>
<pdfbox-to-pageable pages="1"/>
<jps-stream-print output-format="application/postscript"/>
```

Dieses Beispiel lädt ein PDF Dokument, produziert davon ein Pageable Objekt, das von der `jps-stream-print` Pipe mit dem JPS Subsystem von Java in einem PostScript Druckstrom umgewandelt wird.

4.3.17.5. pdfbox-stamp

Versieht PDF Seiten mit einem Stempel bzw. mit Underlays oder Overlays. Dabei werden einseitige PDF-Dateien als Bilder verwendet, in das PDF importiert und entweder unter (Underlay) oder über (Overlay) den bestehenden Inhalt gelegt. Optional kann der eingefügte Inhalt als sogenannte "Optional Content Group" (OCG) hinzugefügt werden, so dass der damit erzeugte Layer ein- und ausgeschalten werden kann.

```
pdfbox-stamp
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
optional-content-group-name	(String, optional) Definiert den Namen der optional "Optional Content Group" (bzw. Layer). Wird dieser Parameter nicht angegeben, wird keine OCG produziert.

Kind-Elemente

Element	Beschreibung
stamp-set+	(1..n) Jedes Stamp Set (Menge von Stempeln) definiert eine Anzahl von Fällen, wobei jeweils pro Seite der erste zutreffende Fall angewendet wird.

[5] http://pdfbox.apache.org

Element	Beschreibung
stamp-set/@condition	(Expression, optional) Hier kann über einen Ausdruck eine Bedingung bestimmt werden, in welchem Fall das Stamp Set zur Anwendung kommt. Beispielsweise kann eine Job Variable abgefragt werden.
stamp-set/stamp+	(1..n) Jeder Stempel referenziert eine PDF-Datei und bestimmt die Positionierung der ersten Seite dieses PDFs auf der Seite. Ausserdem können Konditionen angegeben werden, wann die Stempel-Regel greifen soll.
stamp/@xlink:href	(URI) URI der Stempel-PDF Datei.
stamp/@layer	(enum: under/over) Angabe, ob der Stemp über oder unter den bestehenden Content gelegt werden soll.
stamp/@page-position	(enum: any/first/rest/last/only) Angabe, ob der Stempel verwendet werden soll, je nach Position der Seite. "any" trifft immer zu. "first" gilt nur für die erste Seite. "rest" gilt für alle Seite ausser der ersten. "last" gilt für die letzte Seite. "only" gilt für den Fall, wo es nur eine Seite gibt. Die Reihenfolge der "stamp" Element ist relevant.
stamp/@odd-or-even	(enum: any/odd/even) Angabe, ob der Stempel verwendet werden soll, je nach Seitenzahl. "any" trifft immer zu. "odd" gilt nur ungerade Seiten. "even" gilt nur für gerade Seiten.

Verkettung

E/A	Mögliche Typen
Quelle	PDFBox Dokument (PDDocumentProducer)
Ziel	PDFBox Dokument (PDDocumentConsumer)

Beispiel 4.52. Beispiel für "pdfbox-stamp"

```
<pdfbox-stamp optional-content-group-name="Stamp">
  <stamp-set>
    <stamp page-position="last" xlink:href="stamp1.pdf" layer="over"/>
    <stamp page-position="any" xlink:href="stamp2.pdf" layer="under"/>
  </stamp-set>
  <stamp-set>
    <stamp xlink:href="stamp3.pdf" layer="under"/>
  </stamp-set>
</pdfbox-stamp>
```

Dieses Beispiel zeichnet den Stempel `stamp1.pdf` über den Inhalte der jeweils letzten Seite eines Dokuments. Auf alle anderen Seiten ("any") wird der Stempel `stamp2.pdf` unter den Inhalte gelegt. Und schliesslich wird zusätzlich auf allen Seiten der Stemp `stamp3.pdf` unter den Inhalt gelegt.

4.3.18. Pipes für die Bilder-Verarbeitung

4.3.18.1. rasterize

Rastert mehrseitige Dokumente zur Vollseiten-Bitmaps, z.B. PDF.

```
rasterize
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
rasterizer	(String, optional) Der Raster-Service, welcher zur Rasterung benützt werden soll.
pages	(Page Range, optional) Ermöglicht die Einschränkung auf ein Subset von Seiten des eingehenden Dokuments. Beispiele: "1" verarbeitet nur die erste Seite, "1,3" verarbeitet die Seiten 1 und 3, "3-5" verarbeitet die Seiten 3 bis 5. Es ist also möglich mehrere Einschränkungen komma-separiert anzugeben.

Parameter	Beschreibung
image-format	(MIME Typ, optional) Der MIME Typ des zu generierenden Bitmap-Formats. Standardwert ist "image/tiff". Nur relevant, wenn eine Bitmap-Datei produziert wird. Wird ein Format gewählt, welches nur eine Seite unterstützt, z.B. PNG oder JPEG, wird bei mehreren Seiten eine ZIP Datei erzeugt, welche pro Seite eine Datei enthält.
compression	(String, optional) Der Kompressions-Typ, der bei der Produktion einer Bitmap-Datei verwendet werden soll. Die möglichen Werte hängen vom gewählten Ausgabeformat ab. Mögliche Werte (nicht abschliessend): "CCITT T.4", "CCITT T.6", "PackBits", "LZW", "JPEG", "ZLib", "None". Standardwerte bei Bi-Level (1bit) Bildern is "CCITT T.6", anderenfalls "PackBits".
quality	(Float zwischen 0.0 und 1.0, optional) Bestimmt die Ausgabequalität des Bildes. 1.0 bedeutet maximale Qualität möglicherweise auf Kosten der Verarbeitungsgeschwindigkeit.
bits-per-pixel	(Ganzzahl, optional) Anzahl Bits pro Pixel, z.B. 1, 8 oder 24. Standardwert: 1 für Bi-Level.
resolution	(Ganzzahl, optional) Auflösung der Bilder in Pixel pro Zoll (dpi). Standardwert: 300.

Verkettung

E/A	Mögliche Typen
Quelle	Byte-Strom (InputStreamProducer)
Ziel	Byte-Strom (ByteStreamConsumer) oder ImageProducer

Beispiel 4.53. Beispiel für "rasterize"

```
<stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
<rasterize resolution="300" bits-per-pixel="8"/>
<bitmaps-to-pcl5 resolution="300" quality="0.9" structure="structure"/>
<store-stream expiration="P2D"/>
```

Dieses Beispiel produziert für alle Seiten der PDF-Datei ein Vollseiten-Bitmap bei 300dpi und konvertiert diese anschliessend in einen PCL5 Druckstrom.

Beispiel 4.54. Weiteres Beispiel für "rasterize"

```
<stream source="{$source}" format="application/pdf"/>
<rasterize image-format="image/tiff" resolution="200"/>
<write-to-file file="{$target-file}"/>
```

Dieses Beispiel konvert das eingehende PDF in eine mehrseitige TIFF Datei bei 200dpi und speichert diese auf dem lokalen File System ab.

4.3.18.2. bitmaps-to-stream

Konvertiert eine Serie von Bitmaps in eine Bild-Datei.

```
bitmaps-to-stream
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
pages	(Page Range, optional) Ermöglicht die Einschränkung auf ein Subset von Seiten des eingehenden Dokuments. Beispiele: "1" verarbeitet nur die erste Seite, "1,3" verarbeitet die Seiten 1 und 3, "3-5" verarbeitet die Seiten 3 bis 5. Es ist also möglich mehrere Einschränkungen komma-separiert anzugeben.
image-format	(MIME Typ, optional) Der MIME Typ des zu generierenden Bitmap-Formats. Standardwert ist "image/tiff". Nur relevant, wenn eine Bitmap-Datei produziert wird. Wird ein Format

Parameter	Beschreibung
	gewählt, welches nur eine Seite unterstützt, z.B. PNG oder JPEG, wird bei mehreren Seiten eine ZIP Datei erzeugt, welche pro Seite eine Datei enthält.
compression	(String, optional) Der Kompressions-Typ, der bei der Produktion einer Bitmap-Datei verwendet werden soll. Die möglichen Werte hängen vom gewählten Ausgabeformat ab. Mögliche Werte (nicht abschliessend): "CCITT T.4", "CCITT T.6", "PackBits", "LZW", "JPEG", "ZLib", "None". Standardwerte bei Bi-Level (1bit) Bildern is "CCITT T.6", anderenfalls "PackBits".
resolution	(Ganzzahl, optional) Auflösung der Bilder in Pixel pro Zoll (dpi). Standardwert: 300.

Verkettung

E/A	Mögliche Typen
Quelle	ImageProducer
Ziel	Byte-Strom (ByteStreamConsumer)

Beispiel 4.55. Beispiel für "rasterize"

```
<stream source="{$source}" format="application/pdf"/>
<rasterize resolution="200" bits-per-pixel="8"/>
<bitmaps-to-stream image-format="image/tiff"/>
<write-to-file file="{$target-file}"/>
```

Dieses Beispiel konvert das eingehende PDF in eine mehrseitige TIFF Datei bei 200dpi und mit Graustufen und speichert diese auf dem lokalen File System ab.

4.3.19. Pipes für die PCL 5 Verarbeitung

4.3.19.1. bitmaps-to-pcl5

Konvertiert eine Serie von Vollseiten-Bitmaps in einen PCL 5 Druckstrom. Diese Pipe kann damit z.B. zum gerasterten Druck von PDF Dokumenten benützt werden.

```
bitmaps-to-pcl5
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
resolution	(Integer, optional) Die gewünschte Auflösung in Pixel pro Zoll (dpi). Standardwert: 600 dpi.
quality	(Float zwischen 0.0 und 1.0, optional) Bestimmt die Verarbeitungsqualität der Ausgabe. 1.0 bedeutet maximale Qualität möglicherweise auf Kosten der Verarbeitungsgeschwindigkeit. Die Einstellung beeinflusst in erster Linie das Dithering (Graustufenabbildung).
printer-name	(String, Expr, optional) Bestimmt den Drucker, für den der PCL Datenstrom erzeugt werden soll. Die dahinter steckende Konfiguration bestimmt z.B. die Mediensteuerung basierend auf der Dokumentenstruktur. Fehlt dieses Attribut, wird die Variable "printer-name" konsultiert. Fehlt auch diese, werden Standardwerte angenommen.
structure	(String, optional) Gibt an, von welcher Variable die Struktur-Informationen für das Dokument zu nehmen sind. Diese wird z.B. für die Mediensteuerung benötigt.

Verkettung

E/A	Mögliche Typen
Quelle	ImageProducer
Ziel	Byte Stream

Beispiel 4.56. Beispiel für "bitmaps-to-pcl5"

```
<stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
<rasterize resolution="300" bits-per-pixel="8"/>
<bitmaps-to-pcl5 resolution="300" quality="0.9" structure="structure"/>
<store-stream expiration="P2D"/>
<print/>
```

In diesem Beispiel wird eine PDF Datei auf 300 dpi auf Graustufen gerastert, die einzelnen Vollseitenbitmaps in einen PCL 5 Druckstrom eingebettet, für Forensik-Zwecke 2 Tage aufbewahrt und schliesslich auf den eingestellten Drucker gedruckt (Annahme: Variable "printer-name" wurde anderswo gesetzt).

Der von dieser Pipe generierte PCL 5 Druckstrom unterstützt zur Zeit ausschliesslich monochrom (1 Bit). Farbdruck ist hiermit nicht möglich. Farben und Graustufen werden über Dithering ein 1 Bit Bitmap umgerechnet.

4.3.19.1.1. Unterstütze Features

- Monochrom PCL 5 (1 Bit, schwarz/weiss)
- PJL/JCL Wrapper: hauptsächlich zur Steuerung der Auflösung. Aber es können auch beliebige Einstellungen über die Dokumentenstruktur eingefügt werden.
- Mediensteuerung über Druckerkonfigurationen (Eingabefach, Ausgabefach, Duplex-Druck).

4.3.20. Pipes für die PCL 6 Verarbeitung

4.3.20.1. bitmaps-to-pcl6

Konvertiert eine Serie von Vollseiten-Bitmaps in einen PCL 6 Druckstrom. Diese Pipe kann damit z.B. zum gerasterten Druck von PDF Dokumenten benützt werden. PCL XL wird hier als Synonym für PCL 6 Enhanced verwendet.

```
bitmaps-to-pcl6
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
resolution	(Integer, optional) Die gewünschte Auflösung in Pixel pro Zoll (dpi). Standardwert: 600 dpi.
printer-name	(String, Expr, optional) Bestimmt den Drucker, für den der PCL 6 Datenstrom erzeugt werden soll. Die dahinter steckende Konfiguration bestimmt z.B. die Mediensteuerung basierend auf der Dokumentenstruktur. Fehlt dieses Attribut, wird die Variable "printer-name" konsultiert. Fehlt auch diese, werden Standardwerte angenommen.
structure	(String, optional) Gibt an, von welcher Variable die Struktur-Informationen für das Dokument zu nehmen sind. Diese wird z.B. für die Mediensteuerung benötigt.

Verkettung

E/A	Mögliche Typen
Quelle	ImageProducer
Ziel	Byte Stream

Beispiel 4.57. Beispiel für "bitmaps-to-pcl6"

```
<stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
<rasterize resolution="300" bits-per-pixel="24"/>
<bitmaps-to-pcl6 resolution="300" structure="structure"/>
<store-stream expiration="P2D"/>
<print/>
```

In diesem Beispiel wird eine PDF Datei auf 300 dpi auf RGB gerastert, die einzelnen Vollseitenbitmaps in einen PCL 6 Druckstrom eingebettet, für Forensik-Zwecke 2 Tage aufbewahrt und schliesslich auf den eingestellten Drucker gedruckt (Annahme: Variable "printer-name" wurde anderswo gesetzt).

Der von dieser Pipe generierte PCL 6 Druckstrom unterstützt zur Zeit ausschliesslich RGB (24 Bit).

4.3.20.1.1. Unterstütze Features

- PCL 6 mit RGB Farben (24 Bit)
- PJL/JCL Wrapper: hauptsächlich zur Steuerung der Auflösung. Aber es können auch beliebige Einstellungen über die Dokumentenstruktur eingefügt werden.
- Mediensteuerung über Druckerkonfigurationen (Eingabefach, Ausgabefach, Duplex-Druck).

4.3.21. Pipes für das PostScript Post-Processing

4.3.21.1. ps-media-selection

Erlaubt ein regel-basiertes Einschiessen von PostScript Druckersteuerungs-Kommandos für die Mediensteuerung bzw. Schachtsteuerung.

```
ps-media-selection (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
printer-name (optional)	(Expr) Der WOD-interne Name des Druckers. Standardwert: "`{$printer-name}`"

Verkettung

E/A	Mögliche Typen
Quelle	Byte Strom (application/postscript)
Ziel	Byte Strom (application/postscript)

Beispiel 4.58. Beispiel für "ps-media-selection"

```
<apache-fop-formatter output-format="application/postscript"/>
<ps-media-selection printer-name="Printer007"/>
<apply-ppd model-name="HP Universal Printing PS"/>
<print printer-name="Printer007"/>
```

Dieses Beispiel appliziert die Mediensteuerung für den Drucker "Printer007" auf den von Apache FOP generierten PostScript Druckstrom. Anschliessend wird eine PPD appliziert und das Resultat schlussendlich gedruckt.

Das 'printer-name' Attribut kann weggelassen werden, wenn der Job ein Attribut namens "printer-name" hat. `<ps-media-selection/>` entspricht `<ps-media-selection printer-name="{$printer-name}"/>`

Dieser Ansatz für die PostScript-Verarbeitung wird in Abschnitt 6.4, „PostScript Workflow“ näher beschrieben.

4.3.21.2. apply-ppd

Ersetzt DSC Features in einem PostScript Druckstrom mit Kommandos aus einer PPD (PostScript Printer Definition).

```
apply-ppd (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
printer-name (optional)	(Expr) Der WOD-interne Name des Druckers.

Parameter	Beschreibung
	Standardwert: "`{$printer-name}`"
model-name (optional)	(Expr) Der Modell-Name des Druckers, wie er im "*ModelName" Eintrag in der PPD zu finden ist.
src (optional)	(URI, Expr) Gibt eine URI an, von woher die zu benutzende PPD geladen werden kann.
fail-on-missing-ppd (optional)	(boolean) "true" läst einen Fehler aus, wenn keine passende PPD gefunden werden konnte. Bei "false" werden Fehler ignoriert. Standardwert: "`true`"
inject-jcl (optional)	("auto" oder "disable") "auto" schiesst JCL Kommandos ein, falls die ausgewählte PPD Einträge zu JCL enthält (JCLBegin/JCLToPSInterpreter/JCLEnd) und die Konfiguration des Zieldruckers JCL nicht explizit abschaltet. "disable" erzeugt keine JCL Kommandos. Standardwert: "`auto`"

Verkettung

E/A	Mögliche Typen
Quelle	Byte Strom (application/postscript)
Ziel	Byte Strom (application/postscript)

Beispiel 4.59. Beispiel für "apply-ppd"

```
<apply-ppd src="file:///C:/PPDs/hpcu155s.ppd" inject-jcl="auto"/>
```

Dieses Beispiel appliziert die PPD "hpcu155s.ppd" auf den eingehenden PostScript Druckstrom.

Die 3 Parameter "printer-name", "model-name" und "src" schliessen sich gegenseitig aus. Ist ein "src" Parameter gesetzt, wird dieser verwendet. Ist dieser nicht vorhanden, wird "model-name" konsultiert. Mit dem Modell-Namen wird nach einer im System verfügbaren PPD gesucht. Ist keine der beiden Parameter gesetzt, wird der Parameter "printer-name" ausgelesen. Mit dem Printer-Namen wird nach einer Printer-Konfiguration im System gesucht, welche anzeigt, welche PPD verwendet werden soll. Sind keine der 3 Parameter vorhanden, ist `<apply-ppd/>` equivalent zu `<apply-ppd printer-name="{$printer-name}"/>`.

Falls während des Post-Processings DSC Features in der gewählten PPD nicht gefunden werden konnten, werden diese im Job Log festgehalten.

Weitere Informationen zum Gebrauch von PPDs können unter Abschnitt 6.4, „PostScript Workflow“ nachgelesen werden.

Wird der hier verarbeitete PostScript-Strom anschliessend an einen Drucker geschickt, ist es u.U. notwendig Kommandos mitzuschicken, welche den Drucker auf PostScript-Betrieb umschaltet. Dies geschiet über eine JCL (Job Control Language). Die Kommandos hierfür kommen aus der PPD. In den meisten Fällen wird hierfür die PJL (Printer Job Language) benützt. Dieses Feature lässt sich über das `inject-jcl` Attribut abschalten.

4.3.21.3. ps-apply-structure

Die Pipe ist ein Post-Processor für PostScript-Druckströme, welcher Informationen aus einer Dokumentenstruktur auf den Druckstrom anwenden kann. Beispielsweise ist es damit möglich eine Mediensteuerung umzusetzen.

```
ps-apply-structure (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
structure	(String) Der Name der Variable, welche die Dokumentenstruktur enthält, die auf diesen Druckstrom angewendet werden soll.

Verkettung

E/A	Mögliche Typen
Quelle	Byte Strom (application/postscript)
Ziel	Byte Strom (application/postscript)

Beispiel 4.60. Beispiel für "ps-apply-structure"

```
<stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
<exec dir="D:/Temp" executable="{$ghostscript.exe}" output-format="application/postscript">
  <arg>-dSAFER</arg>
  <arg>-dBATCH</arg>
  <arg>-dNOPAUSE</arg>
  <arg>-q</arg>
  <arg>-sstdout=%stderr</arg>
  <arg>-sDEVICE=ps2write</arg>
  <arg>-dEPSCrop</arg>
  <arg>-sOutputFile=-</arg> <!-- output to stdout -->
  <arg>-</arg> <!-- input from stdin -->
</exec>
<ps-apply-structure structure="structure"/>
<ps-media-selection/>
<apply-ppd inject-jcl="auto"/>
<store-stream expiration="P2D"/>
<print/>
```

Dieses Beispiel benützt GhostScript um eine PDF Datei nach PostScript zu konvertieren. Dem Resultat wird die in der Variable "structure" (anderswo aufgebaut) angewandt, die Mediensteuerung gemäss dem Drucker aus der Variable "printer-name" und schliesslich die PPD angewandt. Das Resultat wird zu Forensikzwecken für 2 Tage gespeichert und der resultierende PostScript Druckstrom auf den gewählten Drucker gedruckt.

GhostScript[6] ist nicht Teil von World Of Documents. Soll GhostScript zusammen mit World Of Documents benützt werden, muss im Einzelfall abgeklärt werden, ob die Software kommerziell lizenziert werden muss (als Artifex GhostScript), da sie in der Open Source Form unter der AGPL Lizenz (AGPL GhostScript) publiziert ist.

4.3.21.3.1. Unterstützte Features

- Anwendung des "media-name" Attributs auf Ebene "document" (PostScript: Defaults) und "page". Gesetzt wird dabei der "%%PageMedia" DSC Kommentar. Dies kann nachgelagert für die Mediensteuerung verwendet werden.
- Anwendung des "ps:features" (Namespace: `http://www.jeremias-maerki.ch/ns/postscript`) Elements auf allen Struktur-Ebenen. Die Anwendung geschieht hierbei im "Setup" Teil eines DSC-konformen für die Features auf "document" Ebene, und im "PageSetup" Teil für jede Seite für den anderen Ebenen.

Nachfolgend ein Beispiel, wie diese Informationen in die Dokumentenstruktur gelangen können. Es wird die Verwendung der `build-document-structure` Pipe gezeigt.

Beispiel 4.61. Beispiel für den Aufbau von Strukturinformationen für PostScript

```
<build-document-structure target-variable="structure"
  xmlns:regexp="http://exslt.org/regular-expressions">
  <xpath expr="true()">
    <set target="document" name="media-name" value="plain"/>
    <set target="document">
      <features xmlns="http://www.jeremias-maerki.ch/ns/postscript">
        <feature name="*Resolution" value="600dpi"/>
        <feature name="*CAPT" value="Middle"/>
      </features>
    </set>
```

[6] http://ghostscript.com/

```
    <set target="page" name="media-name" value="plain"/>
  </xpath>
  <xpath expr="{regexp:test(string(.), '(\d{2}(\d{10})?\d\>)(\d{27}\+)\x20(\d{9}\>)')}">
    <set target="page" name="media-name" value="esr"/>
  </xpath>
</build-document-structure>
```

TODO: PostScript-Code-Auszüge zeigen

4.3.21.4. bitmaps-to-ps

Konvertiert eine Serie von Vollseiten-Bitmaps in einen PostScript Druckstrom. Diese Pipe kann damit z.B. zum gerasterten Druck von PDF Dokumenten benützt werden.

```
bitmaps-to-bitmaps-to-ps
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
structure	(String, optional) Gibt an, von welcher Variable die Struktur-Informationen für das Dokument zu nehmen sind. Diese wird z.B. für die Mediensteuerung benötigt.

Verkettung

E/A	Mögliche Typen
Quelle	ImageProducer
Ziel	Byte Stream

Beispiel 4.62. Beispiel für "bitmaps-to-ps"

```
<stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
<rasterize resolution="300" bits-per-pixel="24"/>
<bitmaps-to-ps structure="structure"/>
<store-stream expiration="P2D"/>
<print/>
```

In diesem Beispiel wird eine PDF Datei auf 300 dpi auf RGB gerastert, die einzelnen Vollseitenbitmaps in einen PostScript Druckstrom umgewandelt, für Forensik-Zwecke 2 Tage aufbewahrt und schliesslich auf den eingestellten Drucker gedruckt (Annahme: Variable "printer-name" wurde anderswo gesetzt).

Der von dieser Pipe generierte PostScript-Druckstrom unterstützt zur Zeit ausschliesslich Monochrom (1 Bit), Graustufen (8 Bit) und RGB (24 Bit), aber kein CMYK oder andere Farbräume.

4.3.21.4.1. Unterstütze Features

- DSC-konformer PostScript Level 3 Druckstrom
- Einfügen von `%%PageMedia` Kommentaren aus der Dokumentenstruktur im "Defaults" Teil sowie für jede Seite.
- Übernahme des Titels aus dem XMP Paket in der Dokumentenstruktur, falls vorhanden (`%%Title` Kommentar).

4.3.22. Pipes für die Kommunikation mit Druckern

4.3.22.1. lpr

Druckt eine Datei auf eine LPR/LPD Druck-Warteschlange (Port 515).

```
lpr (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
hostname	(Expr) Hostname (oder IP Adresse) des Druckers (oder Print-Servers).
queue	(Expr) Name der Warteschlange (Queue).
source (optional)	(Expr) URI des zu druckenden Dokuments.

Verkettung

E/A	Mögliche Typen
Quelle	Signal, Byte Strom
Ziel	Signal

Beispiel 4.63. Beispiel für "lpr"

```
<lpr hostname="localhost" queue="queue1" source="file:///D:/Temp/printfile.ps"/>
```

Dieses Beispiel druckt die lokale Datei `D:\Temp\printfile.ps` auf die Queue 'queue1' des lokalen Print Servers.

Das 'source' Attribut kann weggelassen werden, wenn die Vorgänger-Pipe einen Byte-Strom produziert.

Da der LPR Client nur 10 verschiedene Source Ports benützen darf (RFC 1179) und geschlossene TCP Verbindungen erst nach einem Timeout wiederverwendet werden können, ist die Anzahl der versendbaren Druckjobs pro Minute relativ klein. Müssen viele Druckjobs in kurzer Zeit abgesetzt werden, sollte LPR nicht verwendet werden.

4.3.22.2. raw-print

Druckt eine Datei über das JetDirect/RAW Protokoll (Port 9100). Dieses Protokoll wird von den meisten Netzwerk-Druckern unterstützt.

```
raw-print (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
hostname	(Expr) Hostname (oder IP Adresse) des Druckers (oder Print-Servers).
port (optional)	(Expr) TCP/IP Port des Druckers. Standardwert: 9100.
inject-pjl	(boolean) Ein Wert 'true' generiert um den Druckjob ein minimales PJL Job Ticket, welches versucht den Drucker in die richtige Emulation (je nach MIME Typ des eingehenden Datenstroms) zu versetzen.

Verkettung

E/A	Mögliche Typen
Quelle	Byte Strom
Ziel	Signal

Beispiel 4.64. Beispiel für "raw-print"

```
<raw-print hostname="NPI41860F"/>
```

Dieses Beispiel druckt den eingehenden Druckdatenstrom auf den Jet-Direct-fähigen Drucker 'NPI41860F'.

Bei aktivierter PJL Unterstützung (`inject-pjl="true"`) werden zur Zeit nur drei PDLs unterstützt: PostScript, PCL und PDF. Beispielsweise generiert ein PostScript Datenstrom (application/postscript) das PJL Kommando: `@PJL ENTER LANGUAGE POSTSCRIPT`

4.3.22.3. jps-print

Druckt eine Datei oder ein java.awt.print.Pageable über das Java Printing System, also über das Betriebssystem.

Zu beachten auf Windows Server: Wird WOD unter "LocalSystem" oder "Network Service" betrieben, kann es u.U. die vorhandenen Drucker über JPS nicht ansprechen. Das kann mit Berechtigungen zu tun haben. Daher kann es in bei Benützung der JPS Services nötig werden, WOD unter einem speziell angelegten User laufen zu lassen. Verwandter Knowledge Base Artikel bei Microsoft: http://support.microsoft.com/kb/324565

```
jps-print (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
printer-name	(Expr) Name des Druckers wie im Betriebssytem konfiguriert. Standard: "{$printer-name}"
job-name	(Expr) Name des Druckjobs, wie er an den Spooler gemeldet wird. Fehlt dieser Parameter wird eine generische Beschreibung des Print Jobs erzeugt.

Verkettung

E/A	Mögliche Typen
Quelle	Byte Strom, java.awt.print.Pageable
Ziel	Signal

Beispiel 4.65. Beispiel für "jps-print"

```
<jps-print printer-name="Brother HL-1250"/>
```

Dieses Beispiel druckt den eingehenden Druckdatenstrom auf den im Betriebssystem eingerichteten Drucker namens "Brother HL-1250".

Die Pipe kann zwei Arten von Input verarbeiten. Einerseits kann ein "Pageable" (z.B. von "apache-fop-format-to-pageable") erstellt werden, wobei das Java Printing System mit AWT dazu verwendet wird ein formatiertes Dokument zu drucken. Dabei ist das Betriebssystem und dessen Druckertreiber dafür verantwortlich, die grafischen Elemente in einen druckbaren Datenstrom (PCL, PostScript etc.) umzuwandeln. Andererseits kann aber auch ein Byte Strom zugeführt werden. In diesem Fall wird dieser direkt an den Drucker weitergeleitet, ohne dass der Druckertreiber des Betriebssystems aktiv wird. So kann beispielsweise ein PostScript Druckstrom direkt auf einen Drucker geleitet werden.

4.3.22.4. print

Druckt eine Datei auf einen Drucker, der über eine Ziel-URI Abschnitt 6.6, „Ziel-URIs für Drucker“) angesprochen wird. Diese Pipe ist grösstenteils protokoll-agnostisch und erlaubt den Druck über verschiedenste Druck-Protokolle, welche einen reinen Datenstrom annehmen. Alternativ zur URI lässt sich ein Drucker auch über seinen Namen, wie er im System konfiguriert (Abschnitt 6.4.2, „Druckerkonfiguration“) ist, ansprechen.

```
print (Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
printer-name	(Expr) Name des Druckers wie in der WOD Konfiguration definiert. Standard: "{$printer-name}"
target	(Expr, URI, optional) Die Target URI des Druckers.

Verkettung

E/A	Mögliche Typen
Quelle	Byte Strom

E/A	Mögliche Typen
Ziel	Signal

Beispiel 4.66. Beispiel für "print"

```
<print printer-name="{$printer-name}"/>
```

Dieses Beispiel druckt den eingehenden Druckdatenstrom auf den Drucker, der in der Variable "printer-name" abgelegt ist.

Wird das Attribut "printer-name" weggelassen, wird genau dieser Ausdruck als Standardwert verwendet. Es wird in diesem Fall also erwartet, dass der Druckername in der Variable "printer-name" zu finden ist.

Beispiel 4.67. Beispiel für "print" mit Target URI

```
<print target="lpr://printer1/queue1"/>
```

Dieses Beispiel druckt den eingehenden Druckdatenstrom über das LPR Protokoll auf den Drucker "printer1" (Hostname), und dort auf die Queue namens "queue1". Für Details zu Printer Target URIs, siehe Abschnitt 6.6, „Ziel-URIs für Drucker“. abgelegt ist.

UNC Namen von Druckern können über die XPath Funktion "printerunc2smburl" (Namespace: http://www.jeremias-maerki.ch/ns/wod/functions) in einem SMB URL konvertiert werden. Beispielsweise wird "{wod:printerunc2smburl('\\SERVER01\PRINTER1')}" zu "smb://SERVER01/PRINTER1".

Beispiel 4.68. Beispiel für "print" mit UNC Namen

```
<variable name="printer" value="\\SERVER01\PRINTER1"/>
<print xmlns:wod="http://www.jeremias-maerki.ch/ns/wod/functions"
    target="wod:printerunc2smburl($printer})}"/>
```

4.3.23. Pipes für Pingen

4.3.23.1. pingen

Versendet ein PDF-Dokument via Pingen[7] (Druckdienstleister).

```
pingen
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
account	(String, Expr) Der Name des Pingen-Kontos. Siehe auch Abschnitt 2.7.26, „JM :: Pingen Account“.
send	(boolean) "true", wenn das Dokument gleich nach dem Upload für den Versand freigegeben werden soll. "false", wenn Dokumente lediglich hochgeladen, aber noch nicht freigegeben werden sollen. Standardwert: `false`
Speed	(Enum, Expr, optional) "priority" für prioritären Versand (A-Post in der Schweiz), "pconomy" für kostenminimierten Versand (B-Post in der Schweiz). Fehlt dieses Attribut werden die Voreinstellungen von Pingen bzw. der Konto-Einstellungen übernommen.

[7] https://www.pingen.com/

Parameter	Beschreibung
color	(Enum, Expr, optional) "grayscale" für Graustufendruck, "color" für Vollfarbendruck und "mixed" für gemischten/optimierten Dokumenteninhalt. Fehlt dieses Attribut werden die Voreinstellungen von Pingen bzw. der Konto-Einstellungen übernommen.
duplex	(Enum, Expr, optional) "simplex" für Simplex-Druck, "duplex" für Duplexdruck. Fehlt dieses Attribut werden die Voreinstellungen von Pingen bzw. der Konto-Einstellungen übernommen.
address-placement	(Enum, Expr, optional) "left" für ein Addressfenster auf der linken Seite, "right" für ein Addressfenster auf der rechten Seite. Fehlt dieses Attribut werden die Voreinstellungen von Pingen bzw. der Konto-Einstellungen übernommen.
fail-on-requirements-failure	(boolean) "true", wenn die Pipe fehlschlagen soll, falls keine direkte Freigabe (siehe "send" Attribut) eingestellt ist und die Platzierungsvorgaben von Pingen nicht eingehalten sind. Ist direkter Versand gewählt, schlägt die Pipe in jedem Fall fehl, wenn die Anforderungen für Pingen nicht eingehalten sind. Standardwert: `true`

Verkettung

E/A	Mögliche Typen
Quelle	InputStream (Byte Stream)
Ziel	Signal

Beispiel 4.69. Beispiel für "pingen"

```
<stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
<pingen account="info@acme.com" color="color" address-placement="right"/>
```

Das Beispiel lädt eine PDF Datei zu Pingen hoch, welche gehalten wird, bis sie über die Pingen-Website freigegeben wird. Es wird explizit Vollfarbendruck und Addressfenster auf der rechten Seite gewählt.

Es gibt auch die Möglichkeit, PDF-Dateien über die `print` Pipe an Pingen zu übermitteln, wenn Pingen wie ein virtueller Drucker (Ziel-URI z.B. `pingen:info@acme.com`) behandelt wird.

4.3.24. Pipes für PEAX

4.3.24.1. deliver-to-peax

Lädt ein Dokument in einen PEAX[8] Briefkasten hoch.

```
deliver-to-peax
(Namespace: http://www.jeremias-maerki.ch/ns/wod/pipeline)
```

Parametrisierung

Parameter	Beschreibung
account	(String, Expr) Der Name des PEAX Client/Delivery Accounts. Siehe auch Abschnitt 2.7.25, „JM :: PEAX Client Account“.
validate-metadata	(boolean) "true", wenn die Delivery Metadaten validiert werden sollen. Standardwert: `false`

Verkettung

E/A	Mögliche Typen
Quelle	InputStream (Byte Stream)
Ziel	Signal

[8] https://www.peax.ch/

Beispiel 4.70. Beispiel für PEAX (Metadaten via XSLT)

```
<stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
<deliver-to-peax account="peax.sender.{$wod.environment}">
  <metadata>
    <parse-xml source="{job:representation(job:main-document(),
      'http://www.jeremias-maerki.ch/ns/wod/generic-invoice/v1')}"/>
    <xslt stylesheet="xslt/wod-invoice2peax-metadata.xsl"/>
  </metadata>
</deliver-to-peax>
```

Das Beispiel lädt eine PDF Datei zu PEAX hoch. Die Delivery Metadaten werden hier direkt aus den Rechnungsdaten einer XML Datei via XSLT erzeugt.

Beispiel 4.71. Beispiel für PEAX (Metadaten direkt)

```
<variable name="peax-id" value="973.5284.7314.56"/>
<stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
<deliver-to-peax-test account="jeremias">
  <invoice xmlns="http://www.jeremias-maerki.ch/ns/peax/metadata">
    <source>UPLOAD</source>
    <language>de</language>
    <receiverPeaxId>{$peax-id}</receiverPeaxId>
    <senderName>Jeremias Märki, Software-Entwicklung und Beratung</senderName>
    <senderAddress1>Lützelmattstrasse 14</senderAddress1>
    <senderZip>6006</senderZip>
    <senderCity>Luzern</senderCity>
    <senderCountryCode>CH</senderCountryCode>
    <senderEmail>info@jeremias-maerki.ch</senderEmail>
    <senderUid>CHE-112.165.861</senderUid>
    <title>Test Rechnung 1</title>
    <invoiceDate>2016-08-16</invoiceDate>
    <invoiceNumber>1234</invoiceNumber>
  </invoice>
</deliver-to-peax-test>
```

Dieses Beispiel lädt eine Rechnung in den eigenen Briefkasten (UPLOAD) mit direkt ausgefüllten Delivery Metadaten. Es ist möglich, Expressions zu benützen ("{$peax-id}").

4.4. XPath-Ausdrücke

WOD benützt an vielen Orten XPath zur Auflösung von dynamischen Werten. Da WOD sehr XML-lastig ist, bietet sich dies an. So werden z.B. in Job Definitionen XPath Ausdrücke (Expressions) verwendet um eine Parametrisierung mit den eingehenden Rohdaten zu erreichen. Die Verwendung in XML entspricht weitestgehend dem Ansatz, der in XSLT verwendet wird.

Unglücklicherweise stehen die hier gelisteten XPath Funktionen innerhalb von XSLT Stylesheets nicht zur Verfügung. XSLT wird innerhalb von WOD über das Java JAXP/TraX API angesprochen, welches über keine Möglichkeit verfügt, JAXP XPathFunctionResolver im TraX Transformer zu registrieren. Auch bietet Apache Xalan-J keine ausreichende Infrastruktur diese Funktionen zu integrieren.

Beispiel 4.72. Beispiel für die Verwendung von Ausdrücken

```
<variable name="firstname" value="Hans"/>
<variable name="lastname" value="Muster"/>
<variable name="subject"
  value="Information für {concat($firstname, ' ', $lastname)}"/>
```

In diesem Beispiel kann man sehen, wie ein XPath Ausruck innerhalb der geschwungenen Klammern {} eingefügt wurde. Auch die Standard XPath Funktion **concat()** wurde hier verwendet zusammen mit 2 Variablen.

4.4.1. In WOD verfügbare XPath-Funktionen

Grundsätzlich stehen sämtliche Standard-Funktionen aus der XPath Spezifikation[9] zur Verfügung. Zusätzlich bietet WOD eine Reihe zusätzlicher Funktionen an.

4.4.1.1. Namespace-Prefixe für Funktionen

Tabelle 4.1. XML Namespaces für XPath Funktionen in WOD

Namespace URI	Üblicher Prefix
http://www.jeremias-maerki.ch/ns/xpath	jm
http://www.jeremias-maerki.ch/ns/wod/functions	wod
http://www.jeremias-maerki.ch/ns/wod/job	job
http://www.jeremias-maerki.ch/ns/wod/pipeline	p

4.4.1.2. Globale Funktionen

boolean jm:ifelse(boolean condition, object trueValue, object falseValue)

Liefert je nach Wert des Parameters *`condition`* entweder *`trueValue`* oder *`falseValue`* zurück.

object jm:value(object value, object defaultValue)

Wenn der Parameter *`value`* nicht leer ist, wird dieser zurückgegeben. Sonst wird *`defaultValue`* zurückgegeben.

string jm:uri(object filename)

Wandelt den Parameter *`filename`* in eine URI (als String) um. Eine einfacher Dateiname wird in eine File URI konvertiert. Eine URI wird einfach als String zurückgegeben.

element jm:getConfiguration(string pid)

Liefert eine OSGi Konfiguration, identifiziert durch dessen PID, als DOM Element (XML) zurück.

element jm:getConfiguration(string pid, string format)

Liefert eine OSGi Konfiguration, identifiziert durch dessen PID, im gewünschten Format zurück. *`format`* darf entweder "compact" (für DOM Element) oder "metatype" (OSGi Metatype XML Format) sein.

Das "compact" Format sieht beispielsweise so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<config location="some:location" pid="ch.jm.test.somepid">
  <age>77</age>
  <name>Leia Organa Solo</name>
  <amounts>
    <value>7.77</value>
    <value>77.77</value>
  </amounts>
  <children>
    <value>Jaina</value>
    <value>Jacen</value>
    <value>Anakin</value>
  </children>
  <alive>true</alive>
  <gender>F</gender>
</config>
```

[9] http://www.w3.org/TR/xpath/#corelib

Das "metatype" Format sieht beispielsweise so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<Object location="some:location" pid="ch.jm.test.somepid">
  <Attribute content="77" name="age"/>
  <Attribute content="Leia Organa Solo" name="name"/>
  <Attribute name="amounts">
    <Value>7.77</Value>
    <Value>77.77</Value>
  </Attribute>
  <Attribute name="children">
    <Value>Jaina</Value>
    <Value>Jacen</Value>
    <Value>Anakin</Value>
  </Attribute>
  <Attribute content="true" name="alive"/>
  <Attribute content="F" name="gender"/>
</Object>
```

string wod:filename(string path)

Gibt den Dateinamen (ohne Pfad) zurück.

string wod:filename(URI uri)

Gibt den Dateinamen einer URI zurück, falls vorhanden **wod:filename('http://localhost/test/1234.html'** gibt `1234.html` zurück. Hat die URI keinen Pfad-Anteil (z.B. bei einer URN), wird einfach der "scheme-specific" Teil zurückgegeben, z.B. ergibt **wod:filename('test:local:1234')** den Wert `local:`1234 zurück.

string wod:identifier(string path)

Gibt den Dateinamen (ohne Dateierweiterung und ohne Pfad) zurück. **wod:identifier('D:\Temp\1234.txt')** gibt 1234 zurück.

string wod:identifier(URI uri)

Gibt den Dateinamen (ohne Dateierweiterung und ohne Pfad) zurück. **wod:identifier('D:\Temp\1234.txt')** gibt 1234 zurück. Hat die URI keinen Pfad-Anteil (z.B. bei einer URN), wird einfach der "scheme-specific" Teil zurückgegeben, z.B. ergibt **wod:filename('test:local:1234')** den Wert `local:`1234 zurück.

string wod:formatNumber(object number, string format)

Formatiert eine Zahl als String mit dem vorgegebenen Format. Das Format ist dasselbe, welches von der Java Klasse `java.text.DecimalFormat`[10] definiert wird.

4.4.1.3. Job-spezifische Funktionen

Folgende Funktionen funktionieren nur im Kontext von Pipelines.

URI job:main-document()

Liefert die URI des Hauptdokuments des aktuellen Jobs zurück.

URI job:representation(URI document)

Liefert die URI der ersten gefunden Dokument-Repräsentation (Rendition) des übergebenen Dokuments zurück. Bitte nur verwenden, falls das Dokument garantiert nur eine Repräsentation hat.

[10] http://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

URI job:representation(URI document, string docType)

Liefert die URI der Dokument-Repräsentation (Rendition) des übergebenen Dokuments zurück, welche dem Dokumententyp (*docType*) entspricht. Der Dokumententyp muss als URI angegeben werden. MIME Types müssen der String "ContentType:" vorangestellt werden, also wird "application/pdf" zu "ContentType:application/pdf".

Die häufigste Anwendung dieser Funktionen ist das Laden einer Repräsentation des aktuellen Jobs:

```
<parse-xml source="{job:representation(job:main-document(),
  'http://www.jeremias-maerki.ch/ns/doctypes/offer')}"/>
```

4.4.1.4. Pipeline-spezifische Funktionen

Folgende Funktionen funktionieren nur im Kontext von Pipelines.

document p:build-dom(URI uri)

Lädt eine XML Datei über eine URI und liefert ein DOM Document zurück.

document p:to-xml(object obj)

Serialisiert ein Objekt nach XML, sofern ein Serialisierer für das Objekt zur Verfügung steht, z.B. für einen Job Objekt. Falls für das Objekt kein spezieller Serialisierer gefunden wird, muss das Objekt das Java Interface `org.apache.xmlgraphics.util.XMLizable` implementieren.

4.4.1.5. Eigene Funktionen

Zusätzliche, eigene Funktionen können durch Exponieren eines `javax.xml.xpath.XPathFunctionResolver` als OSGi Service System-weit zur Verfügung gestellt werden. Hierfür sind Java- und OSGi-Kenntnisse vorausgesetzt.

Kapitel 5. Formatieren

Dieses Kapitel behandelt verschiedene Themen zum Thema Formatieren bzw. Composition mit WOD.

Als Dokumentenproduktions- bzw. Output Management System bietet WOD verschiedene Arten, Dokumente zu produzieren. Es geht in erster Linie um die Formatierung/Composition von Rohdaten in ein visuelles Dokument.

5.1. Formatieren mit XSL-FO

Über die Jahre haben sich verschiedene Druckdatenströme (auf englisch: PDLs, Page Definition Languages) auf dem Markt etabliert. Nachfolgend werden die von WOD in irgendeiner Form unterstützen Formate beleuchtet.

Kapitel 6. Drucken

Dieses Kapitel behandelt verschiedene Themen zum Thema Drucken mit WOD.

Als Dokumentenproduktions- bzw. Output Management System bietet WOD verschiedene Arten, wie Drucker angesprochen werden können. Ebenso unterstützt es verschiedene Druckdatenströme, wie z.B. PostScript und AFP. Je nach Aufgabestellung und Umgebung sind verschiedene Ansätze möglich. Nicht jeder Ansatz ist für jede Situation geeignet. Zum Beispiel muss man zwischen zentralem Massendruck (Grossdrucker) und dezentralem Druck auf Einzelplatz- oder Abteilungsdrucker unterscheiden. Bei dezentralen Druck wird schon näher an der Applikation entschieden, auf welchen Drucker ausgedruckt werden soll. Beim zentralen Druck entscheidet vielfach ein Operator im Druckzentrum, auf welchen Drucker ein anstehender Druckjob geschickt werden soll. Im Massendruck werden häufiger Formate wie AFP oder PostScript eingesetzt, während beim dezentralen Druck häufig PCL, aber ebenfalls auch PostScript zur Anwendung kommt. Der direkte Druck von PDF ist zum Teil aufgrund der fehlenden bzw. komplizierten Mediensteuerung noch nicht so weit verbreitet. Häufig werden PDF Dateien für den Druck nach AFP oder PostScript konvertiert. Die folgenden Abschnitte sollen verschiedene Formate und Protokolle kurz beleuchten und dabei Vor- und Nachteile in verschiedenen Szenarien aufzeigen.

6.1. Druckdatenströme

Über die Jahre haben sich verschiedene Druckdatenströme (auf englisch: PDLs, Page Definition Languages) auf dem Markt etabliert. Nachfolgend werden die von WOD in irgendeiner Form unterstützen Formate beleuchtet.

6.1.1. PDF

PDF wurde ursprünglich von Adobe entwickelt und ist heute ISO Standard. PDF ist besonders stark beim elektronischen Dokumentenaustausch. Speziell die Serie der PDF/A Spezifikationen haben PDF weiteren Schub gegeben. PDF/A definiert ein bestimmtes Subset von PDF und bestimmte Regeln, so dass solche PDF Dateien auch in vielen Jahren noch zuverlässig gelesen und reproduziert werden können. Sie eignen sich deshalb auch besonders als Langzeit-Archiv-Format, wie das "A" im Standard schon andeutet.

Der Nachteil von PDF ist die Vernachlässigung von druck-orientierten Metadaten, also z.B. der Mediensteuerung. Hier wird vielfach angenommen, dass diese Metadaten separat geführt und z.B. via JDF an den Drucker übertragen werden, was allerdings vielfach unpraktisch ist. Teilweise müssen Informationen für die Mediensteuerung regelbasiert aus dem Inhalt "erraten" werden.

Zur Zeit kann PDF in WOD nur eingeschränkt als Druckformat verwendet werden. Es kann ohne Probleme generiert werden, wird dann aber in erster Linie als Vorschauformat und als Dokumentenaustausch-Format verwendet. Es ist allerdings möglich die einzelnen Seiten z.B. über Apache PDFBox oder GhostScript in ein anderes Format zu überführen.

Im Bereich des Dokumentenaustausches seien die barriere-freien Fähigkeiten von PDF zu erwähnen. Hier spielen der Standard PDF/UA, aber auch die "a"-Varianten von PDF/A (z.B. PDF/A-1a) eine Rolle.

6.1.2. PostScript

Wie PDF kommt PostScript aus dem Hause Adobe. Ähnlichkeiten in gewissen Teilen sind nicht zu übersehen, auch wenn PostScript praktisch nur als Druckformat eingesetzt wird. Heutzutage unterstützen praktisch alle netzwerkfähigen Drucker PostScript. Die weitgehenden Möglichkeiten zur Druckersteuerung prädestinieren es nicht nur für den zentralen Druck, sondern auch für den dezentralen Druck. Im zentralen Massendruck wird PostScript allerdings zum Teil als zu langsam angeschaut.

Im Umfeld von WOD wird PostScript aus obigen Gründen als präferiertes Druckformat für den dezentralen Druck angeschaut. Die druck-orientierten Post-Processing Funktionen sind hier zur Zeit am besten ausgeprägt. Gerade die Zusatz-Spezifikation DSC (Document Structuring Conventions) macht PostScript sehr einfach und mit hoher Geschwindigkeit verarbeitbar.

WOD unterstützt das Post-Processing mit Hilfe von PPDs (PostScript Printer Definitions). PPDs sind Dateien, die von Druckerherstellern für ihre verschiedensten PostScript-fähigen Druckern bereitgestellt werden und Informationen darüber enthalten, welche Features ein Drucker unterstützt und wie diese anzusprechen sind. Dies macht es möglich, z.B. die Mediensteuerung in weiten Teilen zu generalisieren und damit eine Vielzahl von verschiedenen Druckern in einem heterogenen Umfeld anzusprechen.

6.1.3. PCL 5

PCL 5 ist eine PDL von Hewlett Packard. Von PCL (Printer Command Language) gibt es verschiedene Versionen, die sich zum Teil sehr unterscheiden. PCL 5 ist hier der Vollständigkeit halber erwähnt, wird aber von WOD nicht speziell unterstützt. PCL 5 kann allerdings z.B. im Zusammenhang mit Apache FOP verwendet werden. Dort ist man allerdings auf den Schwarz/Weiss-Druck eingeschränkt.

Der Einsatz von PCL 5 wird weniger empfohlen, auch wenn das Format von den allermeisten Druckern unterstützt wird. Obwohl PCL 5 einige Fähigkeiten hat, gibt es einige Lücken, weshalb viele Druckertreiber Seiten als Bitmaps rastern und pro Seite einfach ein einziges Bitmap in den PCL Strom einbetten.

6.1.4. PCL 6 (PCL XL)

PCL 6 (auch PCL XL genannt) ist gegenüber PCL 5 eine komplett andere Sprache. Sie kommt ebenfalls von Hewlett Packard. Auch PCL 6 wird von WOD nur eingeschränkt unterstützt.

Der Einsatz von PCL 6 ist allenfalls für den dezentralen Druck eingeschränkt empfehlenswert.

6.1.5. AFP

AFP (Advanced Function Presentation) ist ein von IBM entwickeltes Druckformat, welches insbesondere auf Grossdruckern eingesetzt wird. Es ist insbesondere wegen seiner Performance bei grossen Mengen sehr beliebt. Im dezentralen Druck spielt es praktisch keine Rolle. Seine Stärke liegt darum immer wieder verwendete Elemente von Dokumenten (z.B. Dokumentenköpfe und Logos) einmal im Strom abzulegen, vorzurastern und immer wieder zu verwenden. Es lässt sich auch extrem schnell im Post-Processing verarbeiten.

Die Anforderung mit AFP zu arbeiten kommt meistens aus der bestehenden Umgebung. Für die Masse lohnt der Einsatz von AFP. Häufig werden grosse Dokumentenmengen in AFP aufbereitet und auch so archiviert. Im Fall eines Reprint wird das entsprechende Dokument aus dem Strom extrahiert und vielfach nach PDF konvertiert. Der Nachteil hierbei ist, dass AFP keine Features wie Accessibility unterstützt. Man wird also mit einem AFP-zentrischen Ansatz nie barriere-freie Dokumente anbieten können. Ausserdem gibt es bei der Unterstützung und Interpretation von AFP vielfach nicht unerhebliche Unterschiede zwischen den Herstellern und Tools, so dass sich AFP nur eingeschränkt als Archiv-Format eignet. Weil es aber sehr platzsparend ist, wird dies trotzdem gerne gemacht.

6.1.6. Andere Formate

Neben den oben genannten Formaten wird zum Teil auch mit Multi-Page TIFF (Rasterformat) gearbeitet, speziell im Bereich von Archiv-Reprints. WOD wird in Zukunft die Konversion von TIFF-Seitenfolgen in verschiedene der obigen Druckformate unterstützen. Unterstützung für weitere Druckformate ist mangels Kandidaten für verbreitete Formate nicht geplant. Dies gilt insbesondere auch für XPS von Microsoft.

6.2. Druck-Protokolle

Dokument-Inhalte werden üblicherweise mit einem *Media Type* (auch *Content Type* oder *MIME Type* genannt) näher bezeichnet.

6.2.1. LPR

LPR, das Line Printer Remote Protokoll, ist ein altes und weit verbreitetes Druck-Protokoll, das von den meisten Druckern und Print Servern unterstützt wird. Über einen Queue Name (Warteschlange) können verschiedene Drucker über einen Print Server angesprochen werden. Der Standardport für LPR ist Port 515.

WOD kann einerseits über LPR drucken, aber selbst auch als LPR/LPD Print Server auftreten. Dabei ist es möglich verschiedenen Queues verschiedene Workflows zuzuweisen. Eine Einsatzmöglichkeit hier ist beispielsweise die PDF-Produktion aus beliebigen Applikationen, indem man im Betriebssystem einen PostScript-Druckertreiber aufsetzt, dessen Resultat über LPR an WOD geschickt wird. WOD kann diesen eingehenden PostScript-Strom dann in PDF konvertieren, das Dokument in ein DMS oder Archiv ablegen oder per E-Mail verschicken.

Da der LPR Client nur 10 verschiedene Source Ports benützen darf (RFC 1179) und geschlossene TCP Verbindungen erst nach einem Timeout wiederverwendet werden können, ist die Anzahl der versendbaren Druckjobs pro Minute relativ klein. Müssen viele Druckjobs in kurzer Zeit abgesetzt werden, sollte LPR nicht verwendet werden.

6.2.2. JetDirect/RAW

Das JetDirect (oder RAW oder Print Direct) Protokoll wurde insbesondere von Hewlett Packard eingeführt. Die frühen TCP/IP-fähigen Netzwerkdrucker erlaubten den einfachen Druck über Port 9100. Im einfachsten Fall schickt man einfach den rohen (deshalb "raw") PDL Datenstrom auf den Port. Über die PJL (Printer Job Language) lassen sich über den JetDirect Port einerseits das Format des Datenstroms mitteilen und teilweise sogar Informationen wie z.B. den Druckerstatus oder die unterstützten Druckformate vom Drucker abfragen. Die meisten Netzwerkdrucker unterstützen heute dieses einfache Protokoll.

6.2.3. IPP (Internet Printing Protocol)

IPP ist ein Druckprotokoll basierend auf HTTP 1.1 und verfügt ähnlich wie JetDirect über bidirektionale Fähigkeiten z.B. zur Statusabfrage. Der Standard-Port für IPP ist 631. Die meisten heutigen Netzwerkdrucker unterstützen IPP.

WOD enthält zur Zeit nur Unterstützung für client-seitiges IPP (experimentell).

6.2.4. Druck über die Betriebssystem-Infrastruktur

Drucken über die Betriebssystem-Infrastruktur (Drucker-Treiber) ist eine Möglichkeit, die allenfalls beim dezentralen Druck in Betracht gezogen werden kann. In WOD wird dabei über das Java Printing System gedruckt, was wiederum die Drucker-Treiber des Betriebssystem benützt. WOD ist das verwendete Druckformat nicht bekannt, es sei denn, der Druckertreiber meldet explizit Unterstützung, wie es bei vielen PostScript-Druckern der Fall ist. In der Regel ist es möglich einen beliebigen Datenstrom über den Druckertreiber an den Drucker zu schicken, was diesen Ansatz zu einer Alternative zu LPR und JetDirect macht.

Vom Druck via Betriebssystem mit WOD wird eher abgeraten, da in produktiven Umgebungen unter Microsoft Windows Server teilweise keine genügende Stabilität erreicht werden kann. Das kann so weit gehen, dass WOD als Ganzes mit einer Access Violation abstürzt, weil sich das Java Printing System nicht mit Windows bzw. einem bestimmten Druckertreiber "versteht".

6.2.5. CIFS/SMB

CIFS (Common Internet File System, teilweise auch SMB oder Server Message Block genannt) ist ein von Microsoft für die Windows-Produktfamilie entwickeltes Netzwerk-Protokoll, das neben Dateiaustausch auch den Druck auf freigegebene Drucker ermöglicht.

WOD erlaubt den Druck via CIFS Protokoll auf Windows-Drucker. Es bietet in Microsoft-lastigen Umgebungen eine gute Alternative zum Druck direkt via Betriebssystem, da WOD so von Windows über ein Netzwerk-Protokoll entkoppelt ist (siehe Bemerkung zu Stabilität beim Windows-Druck oben).

6.3. Wahl der Druckstrategie

In Abschnitt 6.1, „Druckdatenströme“ und Abschnitt 6.2, „Druck-Protokolle“ wurden verschiedene Möglichkeiten für den Druck mit WOD erwähnt. Generell macht es Sinn, so wenige verschiedene Technologien wie möglich einzusetzen um die Komplexität und den Testaufwand niedrig zu halten.

Im zentralen Druck wird empfohlen, wo möglich mit PostScript[1] bzw. AFP[2] zu arbeiten. PDF[3] ist dann eine Alternative, wenn keine Mediensteuerung notwendig ist, also immer auf weisses Papier gedruckt wird. Empfohlenes Druckprotokoll, soweit verfügbar, ist LPR[4].

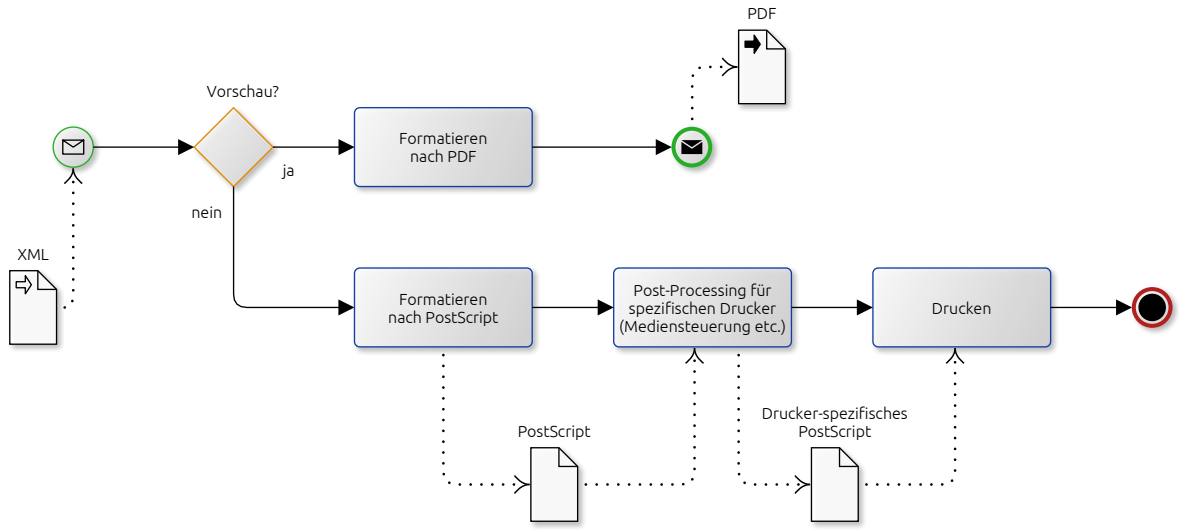
Im dezentralen Druck wird empfohlen, wo möglich mit PostScript zu arbeiten. Die allermeisten netzwerk-fähigen Drucker können heute PostScript[5] verarbeiten. PPD Dateien für diese Drucker sind meist sehr einfach zu beschaffen, was eine weitgehend Drucker-unabhängige Mediensteuerung erlaubt, wo dies notwendig ist. Als Druckprotokoll wird LPR[6] bzw. JetDirect/RAW[7] empfohlen.

Vielfach wird vor dem Druck eine Vorschau eines Dokuments benötigt. WOD erlaubt hier auf sehr einfach Weise die Produktion von PDF[8] Datei für die Vorschau, welche in einem synchronen Aufruf (z.B. HTTP POST[9]) produziert und direkt an die Applikation zurückgegeben werden kann.

6.4. PostScript Workflow

Der PostScript Workflow ist einer der empfohlenen Workflows für WOD, da die allermeisten Drucker PostScript drucken können und Themen wie Mediensteuerung recht einfach abzuhandeln sind. In diesem Abschnitt soll die Umsetzung dieses Workflows mit WOD konkret aufgezeigt werden. Es wird hier beispielhaft angenommen, dass wir als primäre Dokumentenproduktionsmethode XSL-FO mit Apache FOP benützen. Dieser Workflow ist sehr auf das Drucken ausgerichtet. Für die Vorschau-Produktion macht der Workflow keinen Sinn. Dort ist PDF besser geeignet.

Abbildung 6.1. Visualisierung des PostScript Workflow



[1] #printing.formats.postscript
[2] #printing.formats.afp
[3] #printing.formats.pdf
[4] #printing.protocols.lpr
[5] #printing.formats.postscript
[6] #printing.protocols.lpr
[7] #printing.protocols.jetdirect
[8] #printing.formats.pdf
[9] #api-http-submit-sync

WOD bietet für den PostScript-Workflow verschiedene Post-Processing Pipes an, die eine weitgehend Drucker-unabhängige Steuerung erlaubt.

6.4.1. Dezentraler Druck

Schauen wir uns eine mögliche Pipeline für den dezentralen Druck an:

Beispiel 6.1. Beispiel-Pipeline für einen dezentralen PostScript Workflow

```
<?xml version="1.0" encoding="UTF-8"?>
<pipelines xmlns="http://www.jeremias-maerki.ch/ns/wod/pipeline">
  <name>invoice-ps-decentralized</name>
  <assembly>
    <!-- update-job nur zur Illustration -->
    <update-job>
      <attributes>
        <printer-name>PRINTER0071</printer-name>
      </attributes>
    </update-job>

    <!-- hier der eigentliche Workflow -->
    <stream source="{job:representation(job:main-document(),
        'http://www.jeremias-maerki.ch/ns/demo/telco')}"/>
    <parse-xml/>
    <xslt stylesheet="rechnung2fo.xsl"/>
    <apache-fop-formatter output-format="application/postscript"/>
    <ps-media-selection/>
    <apply-ppd/>
    <store-stream expiration="P2D"
        identifier="{wod:identifier($source-uri)}"/>
    <print/>
  </assembly>
</pipeline>
```

Die Pipe `<update-job/>` hier ist nur zur Illustration gedacht. Die Idee dahinter ist, dass der Job ein Attribut "printer-name" von der Applikation mitgeschickt bekommt. Alternativ könnte der Druckername z.B. auch in den XML-Rohdaten hinterlegt sein und mittels XPath ausgelesen werden. Wir nehmen also einfach an, das Attribut "printer-name" enthält den Namen des Druckers, auf den gedruckt werden soll.

Der Ablauf sieht wie folgt aus: Die Eingabedaten für den Job besteht aus einer XML-Datei mit der Format URI "http://www.jeremias-maerki.ch/ns/demo/telco", die als Repräsentation am Hauptdokument des Jobs angehängt ist. Diese XML Datei wird geladen, geparst und mittels dem XSLT Stylesheet "rechnung2fo.xsl" nach XSL-FO transformiert. Der XSL-FO Strom wird an Apache FOP zur Formatierung nach PostScript weitergeleitet. Anschliessend wird der erzeugte PostScript Strom nachbearbeitet, wobei je nach Page Master, der bei der Formatierung einer Seite verwendet wurde, Code für die weitgehend Drucker-unabhängige Medien-Selektion eingefügt wird. Schliesslich wird die zum Drucker gehörige PPD appliziert, wo z.B. bei allfällige "%%IncludeFeature" DSC Kommentare mit dem Code aus der PPD ersetzt werden. Der druckfertige PostScript-Strom wird für 2 Tage (hauptsächlich zu Forensik-Zwecken) im lokalen Dokumenten-Repository gespeichert und anschliessend auf den Zieldrucker geschickt.

Wie wissen nun die Pipes, welchen Drucker wir effektiv ansteuern wollen? Standardmässig benützen die `<ps-media-selection/>`, `<apply-ppd/>` und `<print/>` Pipes den Ausdruck "{$printer-name}" als Standard-Wert für das "printer-name" Attribut. Wir könnten diese Attribute auch explizit setzen, aber das erübrigt sich durch die Verwendung dieser Konvention. Die Pipes wissen also in diesem Beispiel, dass wir auf den Drucker "PRINTER0071" drucken wollen. Damit das alles funktioniert, müssen wir diesen Drucker aber auch in WOD konfigurieren. Das geschieht über folgende Konfigurationsdatei:

6.4.2. Druckerkonfiguration

Beispiel 6.2. Beispielkonfiguration für PRINTER0071

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

```
<printer xmlns="http://www.jeremias-maerki.ch/ns/print">
  <name>PRINTER0071</name>
  <model>HP Universal Printing PS</model>
  <target>lpr://192.168.0.213/queue</target>
  <postscript>
    <media-selection>
      <map>
        <from>
          <dsc-page-media>esr</dsc-page-media>
        </from>
        <to>
          <dsc-input-slot>Tray2</dsc-input-slot>
        </to>
      </map>
      <map>
        <to>
          <dsc-input-slot>Auto</dsc-input-slot>
        </to>
      </map>
    </media-selection>
  </postscript>
</printer>
```

Dem Printer "PRINTER0071" sagen wir, er sei vom Modell "HP Universal Printing PS". Das ist zwar kein eigentliches Druckermodell, aber HP liefert für verschiedene ihrer Laserdrucker eine PPD mit diesem Namen aus. Damit haben wir für die meisten HP Drucker eine einheitliche PPD, welche für die meisten Zwecke ausreicht.

Weiter definiert die Konfigurationsdatei eine Target URI für den Drucker. Wir können sehen, dass dieser Drucker via LPR Protokoll auf der IP Adresse "192.168.0.213" angesprochen werden soll.

Dem folgt der Abschnitt für PostScript Spezifika, allen voran die Mediensteuerung. Die Konfiguration hier sieht so aus, dass sämtliche Seiten, die "esr" als "Page Media" (DSC Kommentar: `"%%PageMedia: esr"`) spezifiziert haben, auf den Input Slot "Tray2" gedruckt werden sollen. `<dsc-input-slot/>` produziert im PostScript hierfür das Kommando `"%%IncludeFeature: *InputSlot Tray"`. Ist die "Page Media" nicht "esr" (also der Fallback), wird der Input Slot "Auto" verwendet.

Ein "ESR" (Einzahlungsschein) ist ein üblicherweise vorgedrucktes Formular, welches in der Schweiz für Zahlungen benützt wird. Der "ESR" ist der Hauptgrund, weshalb das Thema Mediensteuerung gerade in der Schweiz so wichtig ist. Die meisten Drucker können heute ESR nicht mit genügender Qualität auf weisses Papier drucken, zumal zusätzlich eine Perforierung im Papier benötigt wird.

Natürlich könnte man in der `<media-selection/>` auch direkt den PostScript Code zur Auswahl des Papierschachtes einfügen, aber das würde die ganze Konfiguration unübersichtlich machen. Der Performance-Penalty für die Applizierung der PPD ist relative gering.

Das Beispiel hier setzt voraus, dass der PostScript Strom den "%%PageMedia" DSC Kommentar enthält. Apache FOP produziert diesen standardmässig nicht. Man kann ihn über das XSLT Stylesheet über PostScript Erweiterungen für XSL-FO im `fo:simple-page-master` definieren:

```
<ps:ps-page-setup-code name="tray-selection">
  <xsl:text>%%PageMedia: esr</xsl:text>
</ps:ps-page-setup-code>
```

Als Alternative kann man die Drucker-Konfiguration so anpassen, dass direkt der FOP-spezifische Kommentar `%FOPSimplePageMaster` verwendet wird. Standardmässig kann eine von FOP generierte PostScript-Seite etwa so aussehen:

```
%%Page: 3 3
%%PageBoundingBox: 0 0 595 842
%%PageHiResBoundingBox: 0 0 595.275 841.889
%%PageResources: (atend)
```

```
%FOPSimplePageMaster: A4-ESR
%%BeginPageSetup
%FOPBeginSetPageDevice
<<
/PageSize [595 842]
/ImagingBBox null
>> setpagedevice
%FOPEndSetPageDevice
[1 0 0 -1 0 841.889] CT
%%EndPageSetup
```

In diesem Fall würde man das Mapping für den Einzahlungsschein (ESR) wie folgt schreiben:

```
<map>
  <from>
    <comment>FOPSimplePageMaster: A4-ESR</comment>
  </from>
  <to>
    <dsc-input-slot>Tray2</dsc-input-slot>
  </to>
</map>
```

Im `<from/>` Element könnte man auch beide Möglichkeiten gleichzeitig nutzen, da die Übereinstimmung gefunden wird, auch wenn nur einer der beiden passt (OR-Verknüpfung). Es ist also möglich, mehrere Kind-Elemente im `<from/>` Element zu platzieren. Um statt einer OR-Verknüpfung eine AND-Verknüpfung zu erreichen kann auf dem `<from/>` Element ein Attribut `mode="and"` gesetzt werden. Komplexe Konditionen sind zur Zeit noch nicht möglich.

Generell ist empfohlen mit `%%PageMedia` zu arbeiten, da dies Applikations-unabhängiger ist.

Es ist möglich mehrere `printer` Elemente in ein `printers` Element einzubetten und gemeinsam zu deployen.

6.4.3. Zentraler Druck

Beim zentralen Druck, wo wir bei der PostScript-Produktion den Zieldrucker noch nicht kennen, würden wir das Mapping für die Mediensteuerung und die Applizierung der PPD weglassen. Allenfalls kann eine spezielle Druckerkonfiguration für den "virtuellen Drucker Druckzentrale" eingerichtet werden, wo man z.B. über die `<media-selection/>` den `%%PageMedia` DSC Kommentar setzen könnte, falls dieser über Regeln ermittelt werden kann. Auf dem Print-Server in der Druckzentrale kann der Operator dann selbst das entsprechende Mapping vom Medientyp auf den Schacht des Zieldruckers vornehmen.

6.5. PDF Workflow

PDF ist ein sehr universelles Dokumentenformat, das sich insbesondere beim elektronischen Austausch von Dokumenten eignet. Wenn es aber um Transaktionsdruck (oder personalisierter Druck) geht, werden pro Dokument vielfach verschiedene Medien bedruckt, was bei der Verwendung von PDF ein Problem darstellt. In der Schweiz, beispielsweise, werden bei Rechnungen einerseits neutrales oder Logo-Papier und Vordrucke für die Einzahlungsscheine bedruckt. Die PDF Spezifikation unterstützt jedoch keine Auszeichnung der einzelnen Seiten, auf was für ein Medium die Seiten gedruckt werden sollen. Das macht einen Druck-Workflow mit PDF zur Herausforderung.

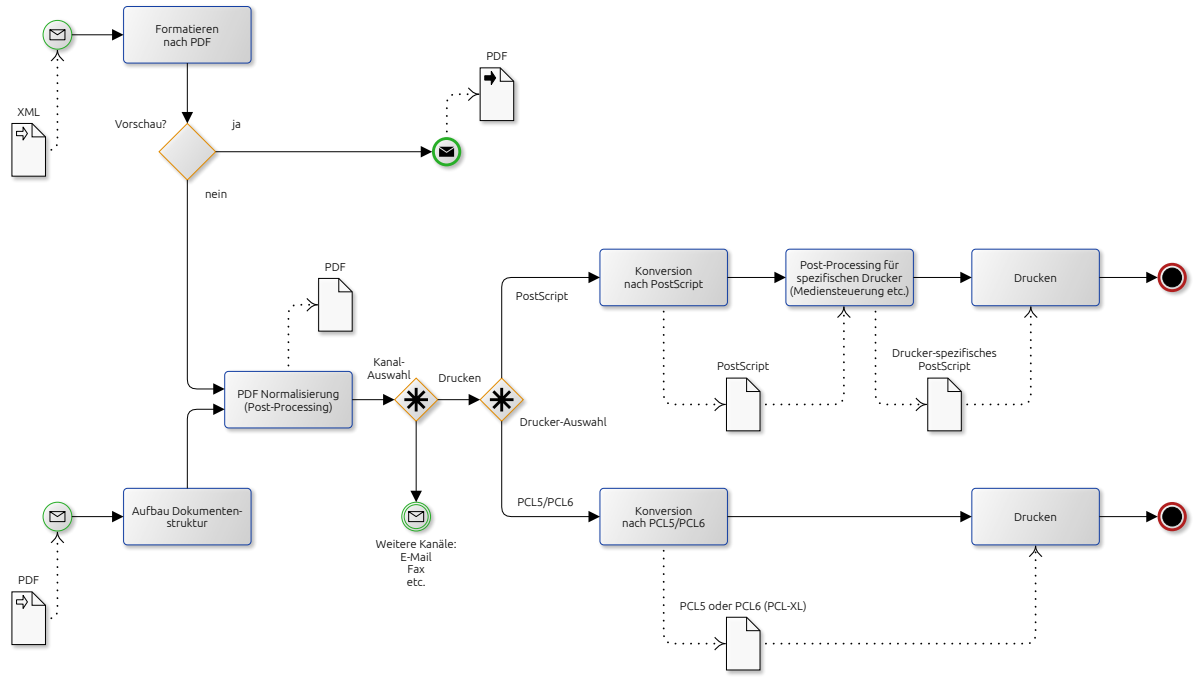
Auswege aus dieser Situation ist die Verwendung von PDF/VT, welche die notwendigen, zusätzlichen Datenstrukturen bereit stellt. Aber PDF/VT ist noch nicht weit verbreitet und schon gar nicht im dezentralen Druck. Da bleiben fast nur noch Parallelstrukturen z.B. in Form einer XML-Datei, welche die fehlenden Informationen zu einer PDF Datei mitführt. WOD bietet genau diesen Ansatz. Natürlich könnte man solche Informationen auch in ein normales PDF schreiben, aber da es sich um proprietäre Erweiterungen handeln würde und potenziell Probleme bei PDF/A-Validatoren verursachen könnten, ist das weniger geeignet. Eine parallele XML-Datei kann sehr einfach, z.B. mit XSLT, für beliebige Zwecke verwendet werden. Beispielsweise lässt sich damit recht einfach eine Steuerdatei für einen Druckdienstleister erstellen, welcher direkt PDF als Druckdatei verarbeiten kann. Dieser benötigt genau aus obigen Gründen ebenfalls solche Parallelstrukturen.

Ein weiterer Grund, weshalb ein PDF-Workflow attraktiv ist, ist die Verfügbarkeit von Komponenten zur Erzeugung und Manipulation von PDF Dateien. Gerade Branchen-Applikationen müssen häufig Tricks anwenden um einzelne Seiten aus einem speziellen Schacht anziehen zu können. Das häufigste Problem ist Windows, welches für einen Druckjob nur einen Papierschacht unterstützt. Vielfach werden dann einfach zwei Druckjobs für ein mehrseitiges Dokument erzeugt, was aber im dezentralen Druck auf einen Arbeitsgruppendrucker, welcher von verschiedenen Benutzern gleichzeitig verwendet wird, zu Problemen führen kann. Und zwar ist die Reihenfolge der Druckjobs nicht garantiert. Im schlimmsten Fall ist es sogar so, dass ein Drucker oder Druckerspooler eingehende Jobs nach Papierschacht sortiert. Das führt dann dazu, dass die einzelnen Seiten manuell wieder richtig zusammengestellt werden müssen. Das ist natürlich eine grosse Fehlerquelle und viel Aufwand.

Wieso also nicht einfach auf irgendeine Weise ein PDF zu erstellen und dies an WOD übergeben, welches sich um die korrekte Druckersteuerung kümmert und daneben gleich viele Ausgabekanäle unterstützen kann (E-Mail mit Attachment, Fax etc.)? Die PDF Dateien müssen gewisse Mindestanforderungen erfüllen. Beispielsweise sollte der Inhalt nicht gerastert sein und alle Schriften sollten eingebettet sein. Bis zu einem gewissen Grad können Schwachpunkte mit entsprechenden Komponenten kompensiert werden. Auch kann so z.B. die PDF/A Konformität hergestellt werden, falls diese nicht gleich vom Erzeuger unterstützt wird.

Die Idee ist nun, dass entweder direkt eine PDF Datei an WOD übergeben wird, oder dass WOD selbst das PDF erstellt. Letzteren Ansatz könnte beispielsweise so aussehen, dass in Windows ein PostScript Druckertreiber eingerichtet wird, bei welchem WOD's LPR Server als Ziel eingerichtet wird. Damit kann aus einer beliebigen Windows-Applikation heraus in WOD hinein gedruckt werden, wobei WOD dann aus dem PostScript Strom (z.B. mit Hilfe von GhostScript) eine PDF-Datei erstellen kann, welche anschliessend weiterverarbeitet wird.

Abbildung 6.2. Visualisierung des PDF Workflow



Nachfolgend werden verschiedene Aspekte das PDF-Ansatzes in WOD näher beleuchtet und mit Beispielen erklärt.

6.5.1. WOD als virtueller Drucker

WOD enthält einen LPR Server, der so aufgesetzt werden kann, dass direkt aus beliebigen Applikationen heraus auf WOD (als virtueller Drucker) gedruckt werden kann. Jeder eingehende Druckjob wird zu einem WOD Job. Es wird empfohlen im Betriebssystem einen PostScript Druckertreiber (z.B. HP Uni-

versal Printing PostScript") zu installieren. Anschliessend kann WOD den PostScript Strom nach PDF konvertieren (z.B. über GhostScript). Alle Information zur Einrichtung können unter Abschnitt 3.2, „LPD/LPR Schnittstelle“ nachgelesen werden.

6.5.2. Formatieren in WOD

WOD kann, z.B. via XSL-FO, auch Dokumente formatieren, basierend auf Rohdaten anstatt extern formatierte Dokumente zu verwenden. Für den PDF Workflow formatieren wir natürlich nach PDF. Das Thema Formatierung ist detailiert unter Kapitel 5, *Formatieren* beschrieben.

6.5.3. PDF Post-Processing

Angenommen, es wird nicht der obige Ansatz mit dem virtuellen Drucker gewählt, und die PDF Dateien werden anderswo produziert, dann kann die Qualität der PDF Dateien durchaus ungenügend sein. Es mag nötig sein, die PDFs nach PDF/A zu konvertieren, Schriften einzubetten, Seiten anzuhängen (z.B. AGBs), Wasserzeichen aufzubringen, eine PDF Datei in mehrere Dokumente (und damit Jobs) aufzusplitten. Da PDF Dateien recht einfach zu manipulieren sind, bietet sich hier wohl die grösste Auswahl an Tools.

6.5.4. Aufbau der Dokumentenstruktur

Da PDF selbst keine standardisierte Form der Mediensteuerung enthält, bietet WOD die Möglichkeit an eine Parallel-Struktur zum eigentlichen Dokument zu führen. Diese Struktur hat 4 Stufen: Dokument, Empfänger, Dokumententeil, Seite.

1. Dokument
2. Empfänger
3. Dokumententeil
4. Seite

Auf jeder Ebene besteht die Möglichkeit Metadaten zu hinterlegen. Beispielsweise kann auf Dokument-Ebene ein XMP Metadaten Paket hinterlegt werden. Auf Empfänger-Ebene kann mittels *CIP4 Common Metadata* die Empfänger-Adresse eines Dokuments abgelegt werden. Die ist gerade in der Zusammenarbeit mit einem externen Druckdienstleister notwendig, welcher u.U. selbst Adressblätter erzeugen können muss, wenn ein Dokument nicht in einem Couvert Platz hat und er dieses auf mehrere Couverts aufteilt. Auf Seiten-Ebene ist die Mediensteuerung angesiedelt. Es lässt sich für jede einzelne Seite angeben, auf was für Papier die Seite gedruckt werden soll.

Beispiel 6.3. Beispiel einer Dokumentenstruktur als XML Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns="http://www.jeremias-maerki.ch/ns/document/structure">
  <properties>
    <Root xmlns="urn:cip4.org:CommonMetadata:CIP4">
      <Metadata>
        <Creator>World Of Documents</Creator>
        <ModificationDate>2015-07-15T10:46:41+02:00</ModificationDate>
      </Metadata>
      <Summary>
        <PageCount>2</PageCount>
        <RecipientCount>1</RecipientCount>
      </Summary>
    </Root>
    <x:xmpmeta xmlns:x="adobe:ns:meta/">
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:xmp="http://ns.adobe.com/xap/1.0/">
        <rdf:Description rdf:about="">
          <dc:format>application/pdf</dc:format>
          <dc:date>2014-12-30T12:53:37+01:00</dc:date>
        </rdf:Description>
```

```
        <rdf:Description rdf:about="">
          <xmp:CreateDate>2014-12-30T12:53:37+01:00</xmp:CreateDate>
          <xmp:CreatorTool>World Of Documents</xmp:CreatorTool>
          <xmp:MetadataDate>2014-12-30T12:53:37+01:00</xmp:MetadataDate>
        </rdf:Description>
      </rdf:RDF>
    </x:xmpmeta>
    <media-name>plain</media-name>
  </properties>
  <recipient>
    <properties>
      <Root xmlns="urn:cip4.org:CommonMetadata:CIP4">
        <Recipient>
          <UniqueId>3095012367</UniqueId>
          <Contact>
            <Person>
              <FirstName>Ursina</FirstName>
              <LastName>Müntener</LastName>
            </Person>
            <Address>
              <AddressLines>
                <Item>Falzigenweg 1</Item>
                <Item>9450 Lüchingen</Item>
              </AddressLines>
              <PostalCode>9450</PostalCode>
              <City>Lüchingen</City>
              <Country>Switzerland</Country>
              <CountryCode>CH</CountryCode>
            </Address>
          </Contact>
        </Recipient>
      </Root>
    </properties>
    <part>
      <page>
        <properties>
          <media-name>plain</media-name>
        </properties>
      </page>
      <page>
        <properties>
          <media-name>esr</media-name>
        </properties>
      </page>
    </part>
  </recipient>
</document>
```

Diese Dokumentenstruktur kann z.B. beim Formatieren mit XSL-FO (nur mit *Apache FOP*) praktisch als Abfallprodukt semi-automatisch aufgebaut werden. Bei der Verarbeitung externer PDF Dateien, kennen wir die Dokumentenstruktur nicht. Sie muss basierend auf Regeln ermittelt werden. Beispielsweise kann für jede Seite der Text extrahiert und z.B. via Regular Expression herausgefunden werden, ob man sich gerade auf der ersten Seite eines Dokuments befindet oder auf was für ein Medium die Seite gedruckt werden soll. Im Fall der Schweizerischen Einzahlungsscheine kann beispielsweise nach der Codierzeile gesucht werden, worauf man den `media-name` z.B. auf "esr" setzt.

6.5.4.1. Struktur-Aufbau mit Apache FOP

Die Apache FOP Pipes erlauben es bei der Formatierung gleich die Dokumentenstruktur aufzubauen und in einer Pipeline-Variable zu speichern (Parameter `structure-target-variable` auf der Pipe-Konfiguration). Ohne spezielle Vorkehrungen produziert die Pipe für ein XSL-FO Dokument (`fo:root`) einen einzigen Empfänger (`recipient`) und pro `fo:page-sequence` einen Dokumententeil (`part`). Ein XMP Metadaten Paket aus `fo:declarations` wird auf Dokumentenstufe aufgenommen. Damit hat die Struktur allerdings erst geringen Wert. Um wirklich aussagekräftige und nützliche Information in die Struktur hineinzubekommen, müssen Erweiterungen im XSL-FO verwendet werden.

WOD Struktur Namespace

`xmlns:struct="http://www.jeremias-maerki.ch/ns/document/structure"`

struct:media

`struct:media` gibt an, auf was für ein Medium eine Seite gedruckt werden soll. Die Werte auf dem `fo:simple-page-master` kommen in der Struktur auf die Seite.

```
<xsl:template match="/">
  <fo:root xsl:use-attribute-sets="defaults" struct:media-name="default">

    <fo:layout-master-set>
      <fo:simple-page-master master-name="esr" struct:media-name="esr"
            page-height="29.7cm" page-width="21cm"
            margin="10mm 10mm 106mm 20mm">
        <fo:region-body margin-top="3.3cm" margin-bottom="1.6em"/>
        <fo:region-before region-name="logo-header" extent="3cm"/>
        <fo:region-after region-name="footer-esr" extent="1.6 * 10pt"/>
      </fo:simple-page-master>
    [..]
```

struct:starts-recipient

`struct:starts-recipient` wird auf `fo:page-sequence` angewendet, womit gesteuert wird, ob die `fo:page-sequence` den Start eines neuen Empfängers anzeigt. Standardmässig wird, wie oben beschrieben, ja nur ein einziger Empfänger pro XSL-FO Dokument erzeugt. Mit diesem Property ist es möglich pro XSL-FO Dokument mehrere Empfänger zu erzeugten.

Für jeden neuen Empfänger wird empfohlen als Kind des entsprechenden `fo:page-sequence` Elements ein *CIP4 Common Metadata* Paket zu erzeugen, welches mindestens die Empfängerdetails enthält.

```
<xsl:template match="rechnung">
  <fo:page-sequence master-reference="rechnung" struct:starts-recipient="true">
     <xsl:apply-templates select="empfaenger" mode="cip4-recipient"/>

     [..]
</xsl:template>

<xsl:template match="empfaenger" mode="cip4-recipient">
  <Root xmlns="urn:cip4.org:CommonMetadata:CIP4">
    <Recipient>
      <UniqueId><xsl:value-of select="@id"/></UniqueId>
      <Contact>
        <Person>
          <FirstName><xsl:value-of select="vorname"/></FirstName>
          <LastName><xsl:value-of select="name"/></LastName>
        </Person>
        <Address>
          <AddressLines>
            <xsl:for-each select="anschrift">
                <Item><xsl:value-of select="."/></Item>
            </xsl:for-each>
            <Item>
              <xsl:value-of select="ort/@plz"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="ort"/>
            </Item>
          </AddressLines>
          <StreetName><xsl:value-of select="strasse"/></StreetName>
          <CivicNumber><xsl:value-of select="hausnummer"/></CivicNumber>
          <PostalCode><xsl:value-of select="ort/@plz"/></PostalCode>
          <City><xsl:value-of select="ort"/></City>
          <Country>Switzerland</Country>
          <CountryCode>CH</CountryCode>
```

```
        </Address>
      </Contact>
    </Recipient>
  </Root>
</xsl:template>
```

struct:*

Alle Attribute nach dem Muster `struct:*` (ausser die oben beschriebenen) werden ohne Präfix als Properties auf der jeweiligen Stufe in die Dokumentenstruktur geschrieben. `struct:logo="abt-tech"` auf `fo:root` würde das Property `logo` auf Stufe Dokument auf "abt-tech" setzen.

XSL-FO zu Struktur Abbildung

XSL-FO Element	Struktur-Stufe
~~fo:root~~ (NYI!)	~~Dokument (`document`)~~
fo:page-sequence	Dokumententeil (`part`), oder Empfänger (`recipient`), wenn `struct:starts-recipient="true"`
fo:simple-page-master	Seite (`page`)

6.5.4.2. Struktur-Aufbau mit durch Seiten-Analyse

Werden in WOD direkt externe PDF Dateien verarbeitet, ist die Dokumenten-Struktur nicht bekannt. Es braucht also Mechnismen, wie diese Struktur basierend auf dem Inhalt und über die Kenntnis der Herkunft der Dokuments abgeleitet werden kann. Wie immer, wenn strukturierte Daten verloren gehen, ist dieser Ansatz beschränkt und je nach Rahmenbedingungen mehr oder weniger zuverlässig. Hier wird nun beschrieben, was WOD diesbezüglich anbietet. Wir beginnen mit einem Beispiel für die Mediensteuerung für eine Rechnung mit (Schweiz-spezifischem) Einzahlungsschein (ESR).

Beispiel 6.4. Ableitung der Struktur eines PDF Dokuments

```
<pipeline>
  <name>pdf-build-structure</name>
  <assembly>
    <stream source="{$pdf-uri}" format="ContentType:application/pdf"/>
    <pdfbox-parse-pdf/>
    <pdfbox-extract-content/>
    <build-document-structure target-variable="structure"
      xmlns:regexp="http://exslt.org/regular-expressions">
      <xpath expr="true()">
        <set target="document" name="media-name" value="plain"/>
        <set target="page" name="media-name" value="plain"/>
      </xpath>
      <xpath expr="{regexp:test(string(.),
            '(\d{2}(\d{10})?\d\>)(\d{27}\+)\x20(\d{9}\>)')}">
        <begin-part/>
        <set target="page" name="media-name" value="esr"/>
        <end-part/>
      </xpath>
    </build-document-structure>
  </assembly>
</pipeline>
```

Die Pipeline in diesem Beispiel tut folgendes: Die PDF Datei wird geladen, durch *Apache PDFBox* geparst und der Inhalt extrahiert. Dies produziert eine XML Datenstruktur im Namespace `http://www.jeremias-maerki.ch/ns/document/content`, welche für jede Seite den Seiteninhalt in einer vereinfachten Form enthält. Dieser Datenstrom wird nun der `build-document-structure` Pipe übergeben, welche für den Aufbau der geschachtelten Dokumentenstruktur zuständig ist. Das `target-variable` Attribut bestimmt, dass die Struktur nach der Erkennung in die Pipeline Variable "structure" geschrieben wird.

Die `build-document-structure` Pipe hier enthält 2 `xpath` Kind-Element. Für alle diese Elemente wird für jede Seite jeweils der XPath Ausdruck ausgeführt. Gibt dieser `true()` als Resultat zurück werden dessen Kind-Elemente ausgeführt. In obigem Beispiel wird also auf Dokument-Stufe der Standard-Medien-

typ "plain" gesetzt, sowie derselbe Wert auf jeder Seite. Im zweiten `xpath` Element existiert eine Einschränkung auf eine Regular Expression (hier das Pattern für eine ESR Codierzeile). Wird das Pattern auf einer Seite entdeckt, wird ein neuer Dokumententeil (`part`) erzeugt (und gleich wieder abgeschlossen), wo bei auf dieser Seite der Medientyp "esr" gesetzt wird. Der Medientyp wird also vom vorherigen `xpath` Element auf Stufe Seite überschrieben.

Die detailierte Beschreibung dieser Pipe ist unter Abschnitt 4.3.4.1, „build-document-structure“ nachzulesen.

6.6. Ziel-URIs für Drucker

In WOD können Drucker über URIs gezielt angesteuert werden. Die URIs enthalten Information über Druck-Protokoll, Drucker-Name, IP-Adressen etc. Hier ein paar Beispiele von Drucker URIs:

- `lpr://printer1/queue1`
- `raw://printer1`
- `jps://BrotherHL1250`
- `smb://server01/BrotherHL1250`
- `ipp://192.168.0.40/ipp/print`

Diese Drucker URIs können insbesondere mit der `<print/>` verwendet werden. Siehe Abschnitt 4.3.22.4, „print“.

6.6.1. LPR Protokoll

Die Syntax für das LPR URI Schema sieht folgendermassen aus:

```
lpr://<hostname>[:<port>]/<queue>[?source=<source>&job-name=<job-name>]
```

Parameter	Beschreibung
hostname	Der Hostname oder die IP-Adresse des Druckspoolers.
port	(optional, Standardwert: 515) Die Port-Nummer für den Druckerspooler.
queue	Der Name der Druckerwarteschlange.
source	(optional) Die Quelle des Druckjobs.
job-name	(optional) Der Name des Druckjobs.

Beispiel 6.5. Beispiel für eine LPR URI

```
lpr://printer01/BINARY_P1
```

Bei Desktop- oder Arbeitsgruppendrucker kann es etwas schwierig sein, den richtigen Namen für die Queue herauszufinden. Manche Drucker akzeptieren einfach alle möglichen Namen. Vielfach funktioniert "print" (z.B. Toshiba MFPs). Drucker von Brother benützen meist "BINARY_P1". Eine Suche auf dem Netz offenbart verschiedene Websites[10], welche LPR Queue Namen auflisten. Ein SNMP Browser kann bei Abfrage des Druckers ebenfalls die möglichen Queue Namen ausgeben.

6.6.2. JetDirect/RAW Protokoll

Die Syntax für das JetDirect/RAW URI Schema sieht folgendermassen aus:

```
raw://<hostname>[:<port>]
```

Parameter	Beschreibung
hostname	Der Hostname oder die IP-Adresse des Druckers.

[10] https://swisscows.ch/?query=printer%20lpr%20queue%20name&region=de-CH&uiLanguage=browser

Parameter	Beschreibung
port	(optional, Standardwert: 9100) Die Port-Nummer für den Drucker.

6.6.3. IPP Protokoll

Die Syntax für das IPP URI Schema sieht folgendermassen aus:

```
ipp://<hostname>[:<port>]/<path>
```

Parameter	Beschreibung
hostname	Der Hostname oder die IP-Adresse des Druckers.
port	(optional, Standardwert: 631) Die Port-Nummer für den Drucker.
path	Der Pfad zum Drucker.

Der Pfad ist von Drucker zu Drucker unterschiedlich. Auf den meisten HP Druckern ist das "/ipp/print". Wird auf einen Linux Print Server (CUPS) gedruckt, wird "/printers/<druckername>" verwendet. Siehe auch die Seite über die CUPS Implementation von IPP[11].

6.6.4. CIFS/SMB Protokoll

Die Syntax für das CIFS/SMB URI Schema sieht folgendermassen aus:

```
smb://[[[<domain>;]<username>[:<password>]@]<server>[:<port>]/<share>
```

Parameter	Beschreibung
domain	(optional) Die Windows Domain.
username	(optional) Der Username.
password	(optional) Das Passwort.
server	Der Server-Name (oder IP-Adresse).
port	(optional, Standardwert: 445) Die Port-Nummer für den Server.
share	Der Share-Name des Druckers.

Es ist zur Zeit leider nicht möglich einen Namen für den Druckjob über dieses URI Schema mitzuliefern.

6.6.5. Druck via Java Printing System

Die Syntax für das JPS URI Schema sieht folgendermassen aus:

```
jps:<printer-name>[?job-name=<job-name>]
```

Parameter	Beschreibung
printer-name	Der Name des Druckers im Betriebssystem.
job-name	(optional) Der Name des Druckjobs.

[11] http://www.cups.org/documentation.php/spec-ipp.html

Kapitel 7. E-Mail

Dieses Kapitel behandelt verschiedene Themen zum Thema E-Mail-Versand mit WOD.

WOD enthält Funktionalitäten zum E-Mail-Versand. Kernstück davon ist eine Pipe, über die Fax-Nachrichten verschickt werden können. Der Versand geschieht intern über Services, welche unabhängig von der Pipe konfiguriert und über einen Service Namen angesteuert werden können.

7.1. Konfiguration der E-Mail Services

Nachfolgend ist beschrieben, wie die verfügbaren Fax Services konfiguriert werden.

7.1.1. E-Mail Simulator

Der E-Mail Simulator Service erlaubt es zu Test-Zwecken E-Mail-Nachrichten auf ein Verzeichnis im lokalen Dateisystem umzuleiten. Die Nachrichten werden im RFC 822 Format (*.eml) gespeichert. So können beispielsweise Workflows getestet werden, ohne dass wirklich E-Mail-Nachrichten verschickt werden müssen. Hier ein Beispiel einer Konfiguration für den E-Mail Simulator:

Beispiel 7.1. Beispiel für die Konfiguration des E-Mail Simulators (Dateiname: ch.jm.email.sender.filesystem-simulator.cfg)

```
name=simulator
target-directory=D:/Temp/_target
```

Mit dieser Konfiguration wird ein E-Mail Sender Service namens "simulator" erstellt, welcher sämtliche E-Mail-Nachrichten in das Verzeichnis `D:\Temp\_target` schreibt (im RFC 822 Format). Der Dateiname einer solchen Konfiguration muss dem Muster `ch.jm.email.sender.filesystem-<name>.cfg` entsprechen.

7.1.2. Versand via SMTP

Der Normalfall beim E-Mail Versand ist die Benützung eines SMTP oder SMTPS Servers. Hier eine Beispielkonfiguration für SMTP:

Beispiel 7.2. Beispiel für die Konfiguration des SMTP Versandes (Dateiname: ch.jm.email.sender.smtp-jeremias-maerki.ch)

```
name=jeremias-maerki.ch

retry-for=PT1M

mail.smtp.host=mail.jeremias-maerki.ch
mail.smtp.user=donteventry
mail.smtp.password=obf:YjY0OnBhc3N3b3Jk
mail.smtp.auth=true
mail.smtp.starttls.enable=true
mail.smtp.starttls.required=true
```

Der Dateiname einer solchen Konfiguration muss dem Muster `ch.jm.email.sender.smtp-<name>.cfg` entsprechen. Mit dieser Konfiguration wird ein E-Mail Versand Service namens "jeremias-maerki.ch" erstellt. Die Konfiguration erzwingt die Authentifizierung mit Username und Passwort ("mail.smtp.auth=true") und verlangt die Verwendung des "STARTTLS" Befehls zur Verschlüsselung der SMTP Verbindung. Ist der SMTP Server temporär nicht verfügbar, wird hier nur während einer Minute versucht, erneut zu versenden.

Parametrisierung

Parameter	Beschreibung
name	(String) Name des E-Mail Versand Service.

Parameter	Beschreibung
retry-for	(String, optional, ISO 8601 Dauer) Die Dauer, während der der Versand des E-Mails versucht wird, wenn der Versand nicht gleich klappt. Standardwert: PT5M (5 Minuten)
minimum-wait-between-retries	(String, optional, ISO 8601 Dauer) Minimale Dauer zwischen erneuten Versuchen, ein E-Mail zu versenden. Standardwert: PT30S (30 Sekunden)
maximum-wait-between-retries	(String, optional, ISO 8601 Dauer) Maximale Dauer zwischen erneuten Versuchen, ein E-Mail zu versenden. Standardwert: PT5M (5 Minuten)
protocol	(String) Definiert das zu verwendende Protokoll: "smtp" (Standard) oder "smtps".
mail.smtp.host	(String) SMTP Hostname oder IP-Adresse.
mail.smtp.user	(String, optional) Username für den SMTP Server.
mail.smtp.password	(String, optional) Passwort für den SMTP Server. Kann verschleiert sein ("obf:....").
mail.smtp.auth	(Boolean, optional) Wird der Wert auf "true" gesetzt, wird Authentifizierung aktiviert.
mail.smtp.starttls.enable	(Boolean, optional) Wird der Wert auf "true" gesetzt, wird das "STARTTLS" Kommando aktiviert um die SMTP Verbindung zu verschlüsseln.
mail.smtp.*	Es gibt viele weitere Parameter, die vom "javax.mail" API zur Verfügung gestellt werden. Die möglichen Parameter hängen von der "javax.mail" Implementierung und dessen Version ab. Verschiedene mehrheitlich allgemeingültige Parameter sind unter http://www.tutorialspoint.com/javamail_api/javamail_api_smtp_servers.htm beschrieben.
mail.smtps.*	Wenn der Parameter "protocol" auf "smtps" gesetzt ist, müssen sämtliche Parameter mit "mail.smtps.*" anstatt "mail.smtp.*" gesetzt sein.

Nachfolgend ein Beispiel für die Einrichtung einer SMTPS Verbindung (SMTP über SSL/TLS). Wichtig dabei ist der "protocol=smtps" Eintrag sowie die Benützung von "mail.smtps.*" statt "mail.smtp.*"

Beispiel 7.3. Beispiel für die Konfiguration des SMTPS Versandes

```
name=sunrise

protocol=smtps
mail.smtps.host=smtp2.sunrise.ch
mail.smtps.user=idontexist@sunrise.ch
mail.smtps.password=obf:YjY0OnBhc3N3b3Jk
```

7.2. Integration in die Dokumentenverarbeitung

Eine spezielle Pipe erlaubt die einfache Integration der E-Mail Services in die Dokumentenverarbeitung mit Pipelines. Hier ein kleines Beispiel:

Beispiel 7.4. Beispiel für den Fax-Versand in einer Pipeline

```
<pipeline>
  <name>mail-pdf</name>
  <assembly>
    <send-mail from="test@jeremias-maerki.ch"
          reply-to="info@jeremias-maerki.ch"
          retry-for="PT0S" service="jeremias-maerki.ch">
      <to>{$email}</to>
      <subject>Ihr Dokument</subject>
      <text>Grüezi, hier ist Ihr Dokument.</text>
```

```
      <html xmlns="http://www.w3.org/1999/xhtml">
        <head>
          <title>HTML Mail Body</title>
          <meta content="application/xhtml+xml;
              charset=UTF-8" http-equiv="Content-Type"/>
        </head>
        <body>
          <p>Grüezi, hier ist Ihr Dokument.</p>
          <p>
            The file is attached:
            {wod:filename(job:representation(job:main-document()))}
          </p>
        </body>
      </html>
      <attachment>{job:representation(job:main-document())}</attachment>
    </send-mail>
  </assembly>
</pipeline>
```

In diesem Beispiel geschieht der E-Mail-Versand über einen E-Mail Service mit dem Namen "jeremias-maerki.ch". Dabei wird Hauptdokuments als Attachment an das E-Mail gehängt. Sowohl Plain-Text als auch HTML Version des E-Mails werden u.a. mit variablen Teilen erzeugt.

Kapitel 8. Fax

Dieses Kapitel behandelt verschiedene Themen zum Thema Fax-Versand mit WOD.

WOD enthält Funktionalitäten zum Fax-Versand. Kernstück davon ist eine Pipe, über die Fax-Nachrichten verschickt werden können. Der Versand geschieht intern über Services, welche unabhängig von der Pipe konfiguriert und über einen Service Namen angesteuert werden können.

8.1. Konfiguration der Fax Services

Nachfolgend ist beschrieben, wie die verfügbaren Fax Services konfiguriert werden.

8.1.1. Fax Simulator

Der Fax Simulator Service erlaubt es zu Test-Zwecken Fax-Nachrichten auf ein Verzeichnis im lokalen Dateisystem umzuleiten. So können beispielsweise Workflows getestet werden, ohne dass wirklich Fax-Nachrichten verschickt werden müssen. Hier ein Beispiel einer Konfiguration für den Fax Simulator:

Beispiel 8.1. Beispiel für die Konfiguration des Fax Simulators (Dateiname: ch.jm.fax.client.filesystem-simulator.cfg)

```
name=simulator
target-directory=D:/Temp/_target
```

Mit dieser Konfiguration wird ein Fax Client Service namens "simulator" erstellt, welcher sämtliche Fax-Nachrichten in das Verzeichnis `D:\Temp\_target` schreibt (üblicherweise im PDF Format). Der Dateiname einer solchen Konfiguration muss dem Muster `ch.jm.fax.client.filesystem-<name>.cfg` entsprechen.

8.1.2. eCall™

eCall™[1] ist ein Online Fax Dienst der Firma Dolphin Systems AG in CH-8832 Wollerau[2]. Der Dienst bietet verschiedene Schnittstellen zum Fax und SMS Versand an. Zur Zeit implementiert WOD lediglich den E-Mail-zu-Fax Gateway, also den Versand von Fax Nachrichten über ein E-Mail an eine ecall.ch Adresse. Die Fax-Nachricht wird dabei als Mail-Attachment mitgeschickt. Damit steht noch keine Funktionalität zur Bestätigung, dass ein Fax angekommen ist, zur Verfügung.

Beispiel 8.2. Beispiel für die Konfiguration des eCall E-Mail-zu-Fax Gateway (Dateiname: ch.jm.fax.client.ecall.email2fax-ecall.cfg)

```
name=ecall

mail-service=jeremias-maerki.ch

from=test@jeremias-maerki.ch
answer-to=info@jeremias-maerki.ch
bcc=info@jeremias-maerki.ch
calling-number=+41413709458
fax-header-id=Jeremias Maerki
fax-header-info=Luetzelmattstr. 14, CH-6006 Luzern

max-retries=0
```

Der Dateiname einer solchen Konfiguration muss dem Muster `ch.jm.fax.client.filesystem-<name>.cfg` entsprechen. Mit dieser Konfiguration wird ein Fax Client Service namens "ecall" erstellt, welcher den E-Mail-zu-Fax Gateway von eCall™ benützt.

[1] https://www.ecall.ch/
[2] http://www.dolphin.ch

Die Beispielkonfiguration hier benützt intern den E-Mail Service mit dem Namen "jeremias-maerki.ch" zum Versand der E-Mail Nachricht mit dem angehängten Dokument. Ausserdem werden u.a. Absender-Fax-Nummer und Info-Zeile gesetzt.

Parametrisierung

Parameter	Beschreibung
name	(String) Name des Fax Service.
mail-service	(String, optional) Name des zu benützenden E-Mail Services für den E-Mail-Versand. Wird dieser Parameter nicht angegeben, wird ein beliebiger E-Mail Service (mit dem höchsten Service Ranking) verwendet. Es wird empfohlen, diesen Parameter immer zu setzen.
from	(String) E-Mail Adresse des Absenders. Diese E-Mail Adresse muss im eCall Profil zum Fax-Versand via E-Mail freigeschalten sein!
answer-to	(String, optional) E-Mail Adresse, an welche Fehlermeldungen beim Fax-Versand geschickt werden. Wird dieser Wert weggelassen, kann man lediglich im eCall Profil das Versand-Log einsehen.
bcc	(String, optional) E-Mail Adresse, an welche sämtliche E-Mails für den Fax-Versand als BCC (Blind Carbon Copy) gesendet werden. Dies dient u.a. zur Fehlersuche oder zur Archivierung.
calling-number	(String, optional, max. 16 Zeichen) Die Absender-Nummer, die dem Empfänger angezeigt werden soll.
fax-header-id	(String, optional, max. 20 Zeichen) Die Absenderkennung des Faxes. Diese Information wird normalerweise in der Headerzeile des Faxes angezeigt und erscheint auch im Display des Faxgerätes.
fax-header-info	(String, optional, max. 50 Zeichen) Die Kopfzeile des Faxes. Diese Information wird normalerweise in der Headerzeile des Faxes angezeigt.
max-retries	(Integer, optional, Werte: 0..5) Die Anzahl der Wiederholungsversuche die Faxnachricht zu senden angeben, welche eCall™ durchführen soll, wenn der Empfänger bereits besetzt oder vorübergehend nicht erreichbar ist.
retries-time-interval-in-min	(Integer, optional, Werte: 0..99) Die Zeit in Minuten zwischen den Wiederholungsversuchen (max-retries) angeben.

8.2. Integration in die Dokumentenverarbeitung

Eine spezielle Pipe erlaubt die einfache Integration der Fax Services in die Dokumentenverarbeitung mit Pipelines. Hier ein kleines Beispiel:

Beispiel 8.3. Beispiel für den Fax-Versand in einer Pipeline

```
<pipeline>
  <name>fax-pdf</name>
  <assembly>
    <stream source="{job:representation(job:main-document())}"
        format="application/pdf"/>
    <send-fax recipient="{$fax}" service="ecall"/>
  </assembly>
</pipeline>
```

In diesem Beispiel geschieht der Fax-Versand über einen Fax-Service mit dem Namen "ecall". Dabei wird die PDF Repräsentation des Hauptdokuments des aktuellen Jobs versandt.

Kapitel 9. E-Rechnung mit PostFinance AG

Dieses Kapitel beschreibt die Anbindung des E-Rechnungssystems von PostFinance AG an WOD. Die Funktionalität umfasst das Hochladen von E-Rechnungen, das Abrufen von Verarbeitungsprotokollen sowie das Abholen von signierten E-Rechnungen zur lokalen Archivierung.

Es ist ratsam, sich vor der Einrichtung mit dem Handbuch E-Rechnung[1] vertraut zu machen. Auch bedingt die Einrichtung eines Vertrag mit PostFinance AG. Details dazu sind auf der Website von PostFinance[2] zu finden.

Es ist wichtig zu wissen, dass WOD keine An- und Abmeldungen von Rechnungsempfängern verwaltet.

9.1. Konfiguration des Rechnungssteller-Kontos

Nach Vertragsabschluss mit PostFinance AG erhält der Rechnungssteller eine "BillerID", einen Benutzernamen und ein Passwort. Mit diesen Informationen wird eine OSGi Konfiguration mit der Factory-PID `ch.jm.wod.ebill.yellowbill.BillerAccount` angelegt.

Beispiel 9.1. Beispiel für die Einrichtung eines Rechnungsstellers

```
biller.id=411010000001234567
biller.name=Acme Corp.
target=ki
username=yb0000000
password=obf:YjY0OmhpZXJnaWJ0c25pY2h0c3p1c2VoZW4=

download.job-type=http://www.jeremias-maerki.ch/ns/demo/yellowbill/signed-invoice
download.scheduler.expression=0 0 * * * ? *

process-protocol.job-type=http://www.jeremias-maerki.ch/ns/demo/yellowbill/process-protocol
process-protocol.scheduler.expression=0 0 * * * ? *
```

Neben den oben erwähnten Konto-Informationen enthält die Konfiguration den Wert "biller.name", welcher aber nur zur Information verwendet wird, damit man bei der Einrichtung mehrere Konten nicht jedes Mal die Biller-ID nachschauen muss.

Weiter werden in der Konfiguration die zwei regelmässigen Tasks für den Download der signierten Rechnungen sowie für den Download der Verarbeitungsprotokolle eingerichtet. "*.job-type" gibt jeweils den Job Typ an, welcher für jede heruntergeladene Datei ausgeführt wird. "*.scheduler.expression" gibt den Cron-kompatiblen Ausdruck an, der den Zeitplan für die Auslösung der zwei Aufgaben bestimmt. Im obigen Beispiel werden die Aufgaben beide stündlich ausgeführt. Die beiden letzten Properties sind optional. Fehlt "process-protocol.scheduler.expression", erfolgt kein automatisierter Download von Verarbeitungsprotokollen.

Das "password" Property kann verschleiert (siehe Abschnitt 12.3, „Passwort-Verschleierung (Obfuscation)“) angegeben werden.

[1] https://www.postfinance.ch/de/biz/prod/eserv/ebill/bill/detail.html
[2] https://www.postfinance.ch/de/biz/prod/eserv/ebill/bill/offer.html

9.2. Download signierter E-Rechnungen

Wie in Abschnitt 9.1, „Konfiguration des Rechnungssteller-Kontos“ bereits angesprochen, kann ein Job zur Verarbeitung heruntergeladener, signierter Rechnungen eingerichtet werden. Im Normalfall wird hierzu eine normale Job-Definition erstellt. Nachfolgend ein vereinfachtes Beispiel, wo die heruntergeladenen E-Rechnungen nur validiert und auf das lokale File System geschrieben werden. In konkreten Umgebungen müssen die Rechnungen MwSt.-konform archiviert werden.

Beispiel 9.2. Beispiel: Job-Definition für die Verarbeitung signierter Rechnungen

```
<job-definition>
  <job-type>http://www.jeremias-maerki.ch/ns/demo/yellowbill/signed-invoice</job-type>
  <description>Demonstrates downloading signed Yellowbill invoices.</description>
  <version>1.0</version>
  <execution>
    <pipeline name="download-yellowbill-signed-invoices"/>
  </execution>

  <pipelines xmlns="http://www.jeremias-maerki.ch/ns/wod/pipeline">
    <pipeline>
      <name>download-yellowbill-signed-invoices</name>
      <assembly>
        <!-- Attribute output-format is necessary since Streamer
             cannot determine the right MIME type, yet. -->
        <stream source="{job:representation(job:main-document())}"
            format="application/signature+xml"/>
        <write-to-file file="D:/yellowbill-{$biller.id}_{$transaction.id}.xml"/>

        <stream source="{job:representation(job:main-document())}"
            format="application/signature+xml"/>
        <yellowbill-extract-pdf/>
        <write-to-file file="D:/yellowbill-{$biller.id}_{$transaction.id}.pdf"/>
      </assembly>
    </pipeline>
  </pipelines>
</job-definition>
```

Für jede heruntergeladene E-Rechnung wird ein eigener Job ausgeführt. Dabei erhält der Job jeweils folgende Job Attribute:

Parameter	**Beschreibung**
incoming-channel {`http://www.jeremias-maerki.ch/ns/wod/job`}	Fixer Wert: "yellowbill"
biller.id	Die Biller ID (Rechnungssteller-Identifikation), z.B. "411010000001234567".
transaction.id	Die Transaction ID der E-Rechnung.
delivery.date	Die Auslieferungsdatum der E-Rechnung, z.B. "2016-02-12".

Das Hauptdokument des Jobs ist jeweils die heruntergeladene, signierte Rechnung (MIME Typ: application/signature+xml).

9.3. Download von Verarbeitungsprotokollen

Ähnlich wie bei den signierten E-Rechnungen können automatisiert Verarbeitungsprotokolle heruntergeladen und weiterverarbeitet werden. Dies ist aber eine optional Funktion, da diese Protokolle auch manuell via E-Rechnung Business Interface eingesehen werden können. Anders als bei den signierten E-Rechnungen, wird pro Ausführung einfach der eingerichtete Job ausgeführt. Der eigentliche Download wird als Teil des Jobs dürchgeführt.

Beispiel 9.3. Beispiel: Job-Definition für Verarbeitsprotokolle

```
<job-definition>
```

```
  <job-type>http://www.jeremias-maerki.ch/ns/demo/yellowbill/process-protocol</job-type>
  <description>Demonstrates what to do with a downloaded process protocols.</description>
  <version>1.0</version>
  <execution>
    <pipeline name="download-yellowbill-process-protocol"/>
  </execution>

  <pipelines xmlns="http://www.jeremias-maerki.ch/ns/wod/pipeline">
    <pipeline>
      <name>download-yellowbill-process-protocol</name>
      <assembly>
        <yellowbill-process-protocol-download biller-id="{$biller.id}"
            create-date="yesterday" archive-data="false"/>
        <write-to-file file="D:/yellowbill-{$biller.id}_protocol.xml"/>
      </assembly>
    </pipeline>
  </pipelines>
</job-definition>
```

Auch dieses Beispiel ist stark vereinfacht und zeigt, wie die Verarbeitungsprotokolle geruntergeladen werden und anschliessend auf das lokale File System geschrieben werden.

Folgende Job-Parameter werden für diesen Job gesetzt:

Parameter	**Beschreibung**
incoming-channel	Fixer Wert: "yellowbill"
{http://www.jeremias-maerki.ch/ns/wod/job}	
biller.id	Die Biller ID (Rechnungssteller-Identifikation), z.B. "411010000001234567".

9.4. Upload von E-Rechnungen

Schliesslich müssen die E-Rechnungen auch an PostFinance AG für die Weiterleitung an die Rechnungsempfänger übermittelt werden. Dies geschieht als Teil einer Pipeline und kann beispielsweise folgendermassen aussehen (vereinfacht):

Beispiel 9.4. Beispiel: Job-Definition für den Upload von E-Rechnungen

```
<pipeline>
  <name>e-invoice.yellowbill</name>
  <assembly>
    <parse-xml source="{job:representation(job:main-document(),
        'http://www.jeremias-maerki.ch/ns/wod/generic-invoice/v1')}"/>
    <xslt stylesheet="http://www.jeremias-maerki.ch/xslt/wod-invoice2yb.xsl"/>
    <validate-xml>
      <schema>resource:/yellowbill/ybInvoice_V2.0.xsd</schema>
    </validate-xml>
    <serialize-xml/>
    <store-stream expiration="P30D" identifier="{wod:identifier($source-uri)}"
        output-format="postfinance:yellowbill-v2"/>
    <add-representation format="postfinance:yellowbill-v2"/>

    <yellowbill-upload
        xml-source="{job:representation(job:main-document(),
            'postfinance:yellowbill-v2')}"
        pdf-source="{$pdf-uri}"/>
    <variable name="delivered" value="true"/>
  </assembly>
</pipeline>
```

Die Login-Informationen werden automatisch aus der Konfiguration mittels Biller ID aus der XML-Datei ermittelt. Deshalb müssen diese hier nicht extra angegeben werden. Dies ermöglicht auch einen Betrieb mit mehreren Mandanten.

Nachfolgend einige Punkte, die bei der Übermittlung von E-Rechnungen an PostFinance zu beachten sind:

- Die hochgeladenen PDFs dürfen nicht signiert sein! Stattdessen muss man PostFinance signieren lassen und die PDFs anschliessend herunterladen und MwSt.-konform archivieren.
- Die hochgeladenen PDFs sollten nicht grösser als 150KB werden. Es gilt also zu beachten, dass keine zu hochauflösenden Logos und Bilder verwendet werden. Idealerweise sollten aber PDF/A-konforme PDFs erzeugt werden, so dass z.B. alle Schriften eingebettet sind.
- Sollen neben einer Rechnung noch zusätzliche Seiten (z.B. Aktionen, AGB-Updates o.ä.) mitgeliefert werden, sind diese besser separat und direkt Kunden zu versenden, um das Rechnungs-PDF nicht unnötig zu vergrössern.
- Ein Einzahlungsschein (ESR) muss im Falle der E-Rechnung nicht zwingend produziert werden. Damit lässt sich Platz einsparen, weil z.B. der OCR-B Font nicht eingebettet werden muss. Es kann sich also lohnen, schon während der Formatierung zu prüfen, auf welche Kanäle eine Rechnung übermittelt wird.

Kapitel 10. Dokumentenversand mit Pingen

Dieses Kapitel beschreibt die Anbindung des Druckdienstleisters Pingen[1] an WOD. Pingen bietet den postalischen Dokumentenversand in die ganze Welt zu sehr attraktiven Konditionen, auch bei kleinen Mengen. Details zum Angebot sind der Pingen Website zu entnehmen.

Soll aus WOD heraus über Pingen gedruckt werden, muss bei Pingen ein Business- oder Integrator-Konto eröffnet werden.

10.1. Konfiguration des Pingen-Kontos

Nach dem Anlegen eines Business- oder Integrator-Kontos bei Pingen erhält man ein API Token, welches in eine OSGi Konfiguration mit der Factory-PID `ch.jm.pingen.Account` eingetragen wird.

Beispiel 10.1. Beispiel für die Einrichtung eines Pingen-Kontos in WOD

```
name=info@acme.com
#environment=production
api.token=01234567890123456aaabbbcccdddeee

default.address-placement=right
default.duplex=simplex
default.speed=priority
```

Das "api.token" Property kann verschleiert (siehe Abschnitt 12.3, „Passwort-Verschleierung (Obfuscation)“) angegeben werden.

10.2. Dokumentenversand

Der Dokumentenversand kann zur Zeit über zwei verschiedene Mechanismen geschehen:

- Pingen als virtueller PDF-Drucker (z.B. `pingen:info@acme.org`) über die `print` Pipe.
- Direkt über die `pingen` Pipe.

Beispiele sind bei der Dokumentation der jeweiligen Pipes zu finden.

10.3. Anpassung der Layouts

Die Verwendung von Pingen bedingt möglicherweise einiger kleiner Anpassungen an den Dokumentenlayouts, da Pingen gewisse Zonen vorgibt, auf die nicht gedruckt werden darf. Beispielsweise gibt es einen fixen Rand von 5mm, Aussparungen für einen DataMatrix Barcode für die Verpackungssteuerung sowie Einschränkungen beim Addressfenster. Werden diese Freihaltezonen nicht eingehalten, wird je nach Einstellungen ein fehlerhaftes Dokument zurückgehalten oder zurückgewiesen. Über die Pingen Website lassen sich PDFs herunterladen, die visualisieren, wo die Fehler bei einem Dokument liegen.

Es ist also beim Druck über Pingen nicht möglich randabfallend zu drucken. Ebenso muss beim Druck von Einzahlungsscheinen (ESR) beachtet werden, dass der Text auf dem Empfangsschein genügend eingerückt ist. Pingen erkennt übrigens Einzahlungsscheine automatisch und druckt diese auf entsprechendes Papier mit Perforation. Das ESR-Formular selbst muss also nicht im PDF vorhanden sein.

Unter der URI `http://www.jeremias-maerki.ch/xslt/pingen2fo.xsl` (Dies ist keine URL! Das Stylesheet befindet sich im Bundle `ch.jm.pingen.jar`) ist ein XSLT Stylesheet verfügbar, welches die reservierten Zonen (Stand Mai 2016) halbtransparent auf eine Seite zeichnen kann. Um dieses Styles-

[1] https://www.pingen.com/

heet ausserhalb der WOD Laufzeitumgebung (z.B. während der Dokumentenentwicklung) verfügbar zu haben, kann der im Bundle eingebette XML Catalog in einen eigenen XML Catalog über folgende Anweisung eingebunden werden:

```
<nextCatalog
  catalog="jar:file:///C:/irgendwo/ch.jm.pingen.jar!/META-INF/catalog/catalog.xml"/>
```

Kapitel 11. Lokalisierung (Localization, L10n)

World Of Documents liefert ein Toolset für die Lokalisierung von Dokumenten mit. Damit lassen sich u.a. mehrsprachige Dokumente z.B. mit XSL-FO recht einfach umsetzen. Im Folgenden werden die Syntax der Übersetzungsdateien sowie des Localization-Filters beschrieben.

Die Informationen in diesem Kapitel sind zum Teil veraltet! Eine Beschreibung eines verbesserten Ansatzes folgt bald.

11.1. Übersetzungsdateien

Die Lokalisierungsmechnismus hier verwendet ein XML Format zur Definition der Übersetzungen. Dies hat den Vorteil, dass sämtliche Unicode Zeichen zur Verfügung stehen und die Dateien einfach aus einem anderen Format generiert werden können, falls nötig.

Beispiel 11.1. Beispiel einer Übersetzungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogue xml:lang="en">
  <message key="title">Worklog report for {person}</message>
  <message key="header.date">Date</message>
  <message key="header.text">Text</message>
  <message key="header.duration">Duration (h)</message>
  <message key="total">Total</message>
</catalogue>
```

Auf `catalogue` wird die Sprache für die Übersetzungen bestimmt. Jedes `message` Element definiert ein Schlüsselwort (key). Der Inhalt ist normalerweise ein einfacher String, kann aber auch Variablen enthalten (z.B. "{person}"). Die detailierte Syntax für die Variablen wird weiter unten beschrieben.

Diese Übersetzungsdatei könnte nun den Namen "mytranslations.xml" heissen. Eine zweite könnte die deutsche Übersetzung (z.B. "de_CH" für Schweizerisches Deutsch) beinhalten. Die Sprache wird dann in den Dateinamen codiert: "mytranslations_de_CH.xml". Gibt es nur diese beiden Übersetzungsdateien, wird aber Französisch angefragt, fällt der Übersetzungsmechanismus auf die Standardsprache zurück, also in diesem Fall Englisch, weil bei dieser Datei die Sprache nicht in den Dateinamen hineincodiert wurde.

11.2. Syntax der Variablen

11.2.1. Einfache Variable

Ein Ausdruck wie "Die Rechnung ist innert {duedays} Tagen zu bezahlen." könnte zum Beispiel nach "Die Rechnung ist innert 30 Tagen zu bezahlen." aufgelöst werden. Die Benützung von Variablen hat den Vorteil, dass ein Übersetzungstext nicht in 3 Teile aufgeteilt werden muss. Bei Verwendung mehrerer Variablen könnte auch je nach Sprache die Reihenfolge der Variablen unterschiedlich sein. Die sprechenden Namen der Variablen helfen ausserdem der Leserlichkeit.

11.2.2. Choice

Um sprachliche Unterschiede bei verschiedenen Zahlen auszugleichen kann "choice" eine kompakte Syntax bieten. Ein Beispiel: "Du hast {amount,choice,0#nichts|0<{amount} Franken|100<mehr als genug}."

Ist die Variable "amount" gleich 0, ist das Resultat "Du hast nichts.". Ist der Wert kleiner 1000, ist das Resultat "Du hast 778 Franken.". Ist der Wert 1000 oder mehr, ist das Resultat "Du hast mehr als genug.".

11.2.3. If

Eine etwas einfachere Art der konditionalen Ausdrücke ist das "if". Ein Beispiel: "Du bist {isGood,if,sehr,nicht so} nett!".

Ist "isGood" ein Boolean Wert und "true", ist das Resultat: "Du bist sehr nett!", und sonst "Du bist nicht so nett!".

11.2.4. Equals

Dies ist eine leichte Variante des "if". Es erzeugt einen Text, je nachdem, ob zwei Werte miteinander übereinstimmen. Ein Beispiel: "{isNew,equals,new,Das ist ja neu!,Hmm\, ziemlich alt!}".

Ist "isNew" gleich "new", ist das Resultat: "Das ist ja neu!", und sonst "Hmm, ziemlich alt!". Zu beachten gilt hier, dass Strings verglichen werden. Ein Zahlenwert von "1.20" ist nicht equivalent zu "1.2"! Ausserdem wurde im Beispiel ein Komma mit einem Backslash "escaped", welches sonst als Feldseparator erkennt würde. Also wird "\," als "," interpretiert.

11.3. Die Syntax des XML Lokalisierungsfilters

Die Übersetzungsfunktionalität beinhaltet einen XML Filter, der die eigentliche Übersetzung von XML Dateien übernimmt. Der Filter überträgt im Wesentlichen sämtlichen XML-Inhalt unverändert, es sei denn, er findet spezielle Steuerelemente aus dem "http://jeremias-maerki.ch/l10n" Namespace, welche hier beschrieben werden. Der empfohlene Namespace Prefix für diesen Namespace ist "l10n".

Neben dem oben beschriebenen Namespace beachtet der Filter ausserdem das "xml:id" Attribut, mit welchem die aktuelle Sprache für einen Ast (oder den ganzen XML Baum) festlegt.

11.3.1. Das "l10n:string" Element

Das "l10n:string" Element enthält als Text-Inhalt den Schlüsselwert für eine Übersetzung. Ein Beispiel: "<l10n:string>title</l10n:string>". Dies würde das Element mit dem Text aus dem Übersetzungstemplate "title" ersetzen.

11.3.2. Das "l10n:*" Wildcard-Element

Hier handelt es sich im Prinzip um eine Vereinfachung von "l10n:string", wo der Template-Name bereits im Elementnamen verschlüsselt werden kann. Dies bringt aber eine leichte Einschränkung auf die möglichen Namen der Templates, weil nur gültige XML Elementnamen verwendet werden können. Dasselbe Beispiel von oben, aber mit dieser kompakten Syntax: "<l10n:title/>"

Ausserdem sind hier zusätzliche Parameter erlaubt. Nehmen wir ein Beispiel von oben. Ein Übersetzungstemplate: "<message key="title">Arbeitsrapport von {person}</message>" Nun kann im XML z.B. folgendes geschrieben werden:

Beispiel 11.2. Beispiel für l10n:* mit Parametern

```
<fo:block><l10n:title person="Maria Bernasconi"/></fo:block>
```

Kapitel 12. Security

Dieses Kapitel behandelt verschiedene Themen zum Thema Security.

12.1. Standard-Einstellungen bezüglich Sicherheit

Bei den Standard- und DocGen-Editionen von WOD sind sämtliche Zugriffsbeschränkungen abgeschalten! Eine entsprechende Meldung wird beim Start auch ins Log geschrieben.

Die Sicherheits-Features von WOD können über das "wod.security.disabled" Framework Property eingeschalten werden. Im mitgelieferten `framework.properties` ist dieses Property auf "true". Dieses muss also auf "false" gestellt werden. Danach ist ein Neustart von WOD notwendig.

Bei abgeschalteter Security wird ein "anonymous" User mit "user" und "admin" Rollen registriert, der auf alle Mandanten im System Zugriff hat.

Zur Zeit sind die Sicherheits-Mechanismen in WOD noch etwas beschränkt. Zumindest der Zugriff via HTTP kann über Filter eingeschränkt werden. WOD baut intern auf Apache Shiro[1] für die Authentifizierung und Authorisierung.

TODO: Shiro-Konfiguration und weitere Sicherheitsmechanismen dokumentieren.

12.2. Security mit Apache Shiro

WOD benützt intern Apache Shiro[2] als Security-Mechanismus. Der grösste Teil der Dokumentation von Apache Shiro gilt also auch für WOD.

Bei Apache Shiro kann man über eine `shiro.ini` Datei das Framework konfigurieren. Die offizielle Dokumentation von Shiro beschreibt die Details. In WOD kann man die `shiro.ini` Datei ins `etc` Verzeichnis ablegen. Dann wird es automatisch verwendet. Bei der Konfigurierung ist es ratsam vorsichtig vorzugehen, damit keine Sicherheitslücken entstehen.

Beispiel 12.1. Beispiel für ein shiro.ini

```
[main]
cacheManager = org.apache.shiro.cache.MemoryConstrainedCacheManager
securityManager.cacheManager = $cacheManager

[urls]
/info/** = noSessionCreation, authcBasic, roles[admin]
/health/** = noSessionCreation, authcBasic, roles[admin]
/javamelody/** = noSessionCreation, authcBasic, roles[admin]
/system/** = authcBasic, roles[admin]
/wod-admin/** = authcBasic, roles[admin]
/wod/** = noSessionCreation, authcBasic
/** = authcBasic
```

Dieses Beispiel aktiviert einen In-Memory Cache und definiert eine Reihe von URLs mit bestimmten Sicherheitsmerkmalen. Die ersten 3 erhalten ein "noSessionCreation", weil diese üblicherweise für einfache HTTP GET Abfragen durch technische Clients abgefragt werden. Allerdings wird dort immer auch die "admin" Rolle verlangt. Bei `/wod-admin` wird zwingend eine Session benötigt, weshalb dort "noSessionCreation" weggelassen wird. Bei `/wod` wird nicht auf eine Rolle geprüft. Stattdessen wird dort überprüft, ob das "tenants" Rollenattribut entsprechend gesetzt ist.

[1] http://shiro.apache.org/
[2] http://shiro.apache.org

12.2.1. OSGi UserAdmin Plug-in

WOD enthält ein Plug-in für Shiro, welches den OSGi UserAdmin Service zur Authentifikation und Authorisierung benützt[3].

Ist dieses Plug-in aktiv, können User und Gruppen/Rollen über das UserAdmin Plug-in in der Apache Felix WebConsole verwaltet werden. Alternativ kann eine Konfiguration mit dem Dateinamen `ch.jm.osgi.useradmin.init.cfg` erstellt werden, um den OSGi UserAdmin Service initial zu befüllen. Wird die Konfiguration mehrfach ausgeführt, werden keine Werte überschrieben. Ein manuell geändertes Passwort wird also nicht zurückgesetzt bzw. überschrieben.

Beispiel 12.2. Beispiel für ein ch.jm.osgi.useradmin.init.cfg

```
user.peter.realname=Peter Tester
user.peter.credential.password=$shiro1$SHA-256$500000$YuXazEUSKkMYeF2Jif/T9Q==$kapREv[..]
user.peter.tenants=*

user.acme.realname=Acme Ltd.
user.acme.tenants=acme
user.acme.technical=true
user.acme.credential.password=$shiro1$SHA-256$500000$GXuIFl+AWJ3K7YY56Nw/gA==$j88cJfB[..]

group.admin.users=peter

group.webconsole.groups=admin
```

Diese Konfiguration definiert einen Benutzer "peter" mit einem Shiro Passwort, das dem Konto zugeordnet ist. Peter kann auf alle Mandanten ("*") zugreifen.

Dann gibt es den technischen Benutzer "acme", auch mit einem Shiro Passwort, welcher Zugriff nur auf den Mandanten "acme" erhält. Das Rollenattribut "technical" ist hier nur Information. Es wird zur Zeit nicht benützt.

Weiter definieren wir eine Gruppe "admin", der wir Peter zuordnen. Benutzer "acme" ist kein Administration und hat deshalb keinen Zugriff auf die Administrations-Interfaces. Und schliesslich fügen wir die "admin" Gruppe der Gruppe "webconsole" hinzu. Dies ist notwendig, damit Administratoren Zugriff auf die Apache Felix WebConsole erhalten, wenn die Sicherheitsmechanismen von WOD eingeschalten sind.

12.2.2. Passwort Hashes

Passwort Hashes für Apache Shiro können entweder mit dessen Kommandozeilen-Tool berechnet werden. Alternative enthält die WOD Admin Konsole ein "Passwort Plug-in", wo man einen Shiro-kompatiblen, sicheren, gesalzenen SHA-256 Passwort-Hash berechnen lassen kann.

12.3. Passwort-Verschleierung (Obfuscation)

Es kann kaum verhindert werden, dass in Konfigurationsdateien an verschiedenen Orten Passwörter hinterlegt werden müssen. Es ist also in einer Umgebung mit höheren Sicherheitsanforderungen ganz wichtig, dass die Zugriffsberechtigungen für die Konfigurationsdateien entsprechend geschützt sind.

Bei der Arbeit am Bildschirm kann es aber doch vorkommen, dass einem jemand über die Schulter schaut. Um die Passwörter in so einem Fall nicht allzu einfach durch eine kleine Unachtsamkeit zu kompromittieren, unterstützt WOD an den meisten Stellen eine Passwort-Verschleierung (Password Obfuscation). Die Passwörter werden dabei auf den ersten Blick unlesbar gemacht, aber **nicht** kryptografisch verschlüsselt. Diese Unterscheidung ist sehr wichtig. Mit dem richtigen Tool können solche Passwörter sehr einfach in Klartext umgewandelt werden.

[3] Bundle: ch.jm.wod.security.realm.useradmin

Passwort-Verschleierung (Obfuscation) ist keine starke Sicherheitsmassnahme!

Verschleierte Passwörter sind am Präfix "obf:" zu erkennen. Um Passwörter zu verschleiern kann das Passwort-Plugin im WOD Admin GUI verwendet werden.

Kapitel 13. Problembehandlung

Dieses Kapitel enthält Tips zum Umgang mit und zur Lösung von Problemen im Zusammenhang mit WOD.

13.1. Es tritt folgender Fehler auf (Fehlermeldung kann leicht abweichen): „java.lang.ClassNotFoundException: org.apache.xpath.jaxp.XPathFactoryImpl not found by [..]“. Wie kann dies vermieden werden?

In `%WOD-HOME%/etc/framework.properties` kann `org.apache.xpath.jaxp` in die Liste der Java Packages des `org.osgi.framework.bootdelegation` Properties aufgenommen werden.

Technischer Hintergrund

Die Routine zur Ermittlung der JAXP XPathFactory ist nicht OSGi-freundlich geschrieben und kann deshalb Probleme verursachen. Details im Internet.[1]

13.2. Was muss ich beachten, wenn ich z.B. bei der PDF-Produktion und manchmal auch beim Druck Logopapier (Vordruck) simulieren muss (Underlay, aber auch Overlay)? Manchmal gibt es Fehler beim Einbetten von PDF, oder die Ausgabe bei PostScript ist fehlerhaft.

Folgende Informationen sind z.T. auch für die normale Benützung von PDF Dateien als Bilder relevant.

Natürlich wäre es am Komfortabelsten, wenn man einfach ein Vollseiten-PDF als Seitenhintergrund aufbringen könnte. Dass dies geht, ist auch das langfristige Zeit von WOD. Aber die Realität weicht noch etwas davon ab. Hier deshalb ein paar Tips & Tricks zum Thema Underlays und Overlays.

Fehler beim Einbetten von PDF bei PDF-Produktion mit FOP

Normalerweise funktioniert das Einbetten von PDF in PDF (PDFBox Images Plug-in von Apache FOP[2] bei der Verwendung von XSL-FO zur Formatierung) recht gut. Beobachtet wurden Probleme bei PDFs, die von MS Word erzeugt wurden. Bei solchen Problem ist es ratsam, das PDF mit 3-Heights™ PDF Analysis & Repair[3] oder GhostScript[4] zu korrigieren. Bei GhostScript kann man folgendes Kommando ausführen:

Beispiel 13.1. Reparatur einer PDF-Datei mit GhostScript

```
gswin64c -dSAFER -dBATCH -dNOPAUSE \
    -sDEVICE=pdfwrite \
    -sOutputFile=MyLogo-Fixed.pdf MyLogo.pdf
```

Dies sollte die meisten Probleme verhindern.

Darstellungsfehler beim Einbetten von PDF bei PostScript-Produktion mit FOP

Apache FOP benützt Apache PDFBox für die Konversion von PDF nach PostScript (über die Java-eigene Java2D Infrastruktur). Allerdings ist die Java2D Unterstützung von FOP bei der PostScript-Generierung nicht vollständig, weshalb Darstellungsfehler auftreten können. Folgende Einstellung kann in der FOP Konfigurationsdatei ausprobiert werden:

[1] http://apache-felix.18485.x6.nabble.com/quot-No-XPathFactory-implementation-found-for-the-object-model-quot-when-using-XPath-td4998502.html
[2] http://xmlgraphics.apache.org/fop/fop-pdf-images.html
[3] http://www.pdf-tools.com/pdf/pdf-analysis-repair-corrupt-files.aspx
[4] http://ghostscript.com/

```
<fop>
  [..]
  <target-resolution>300</target-resolution>
  [..]
  <image-loading>
    <penalty value="-11"
      class="org.apache.xmlgraphics.image.loader.impl.ImageConverterG2D2Bitmap"/>
  </image-loading>
  [..]
</fop>
```

Dies favorisiert die Produktion eines Ganzseiten-Bitmaps gegenüber dem Rendering des PDFs via Java2D (G2D, Graphics2D) Infrastruktur. Allerdings hat das Auswirkungen nicht nur auf die PostScript-Konversion, sondern auch auf alle anderen von FOP unterstützte Ausgabeformate. Dasselbe gilt für folgendes Beispiel:

```
<image-loading>
    <penalty value="INFINITE"
      class="org.apache.fop.render.pdf.pdfbox.ImageConverterPDF2G2D"/>
  </image-loading>
```

Dieses Beispiel deaktiviert das Rendering von PDFs mit Apache PDFBox via Java2D. Aber auch diese Massnahme hat weitreichende Nachteile.

Dies sind Gründe dafür, momentan keine PDF Dateien als Under-/Overlays zu benützen, wenn auch PostScript generiert werden soll. Stattdessen sollte, wenn möglich, SVG benützt werden. Zur Not geht auch ein Ganzseiten-Bitmap einer PDF Seite, die z.B. über folgenden GhostScript-Aufruf manuell erstellt werden kann:

Beispiel 13.2. Konversion einer PDF-Datei nach PNG mit GhostScript

```
gswin64c -dSAFER -dBATCH -dNOPAUSE -sDEVICE=png16m -r300x300 \
  -dMaxBitmap=2147483647 -dTextAlphaBits=4 -dGraphicsAlphaBits=4 \
  -sOutputFile=Background.png Background.pdf
```

Empfohlenes Vorgehen für Under-/Overlays mit Apache FOP

Muss nur PDF erzeugt werden, ist die Verwendung von qualitativ guten PDFs empfohlen.

Bei Produktion von verschiedenen Ausgabeformaten ist SVG vorzuziehen. Einziger Nachteil: SVG Bilder können von Apache FOP zur Zeit nicht als sogenannte Form XObjects eingebettet werden, weshalb sie bei jedem Auftreten des Bildes jeweils neu gezeichnet werden, was die Ausgabedatei im schlimmsten Fall wesentlich grösser machen kann.

Grundsätzlich kann auch die Verwendung des Abschnitt 6.5, „PDF Workflow “ in Betracht gezogen werden, was natürlich weitreichende Konsequenzen haben kann. In diesem Fall ist es für ganzseitige Under-/Overlays auch denkbar, das Aufbringen als Post-Processing Schritt durchzuführen.

Funktioniert alles nicht, kann ein Bitmap (je nach Bild-Typ PNG oder JPEG) verwendet werden. Aber auch dieser Ansatz kann die Ausgabedateien wesentlich vergrössern und die Verarbeitungs-/Druckperformance negativ beeinflussen. Dafür ist das Funktionieren praktisch garantiert.

13.3. Wie konvertiere ich PDFs am besten nach SVG?

Inkscape[5] kann PDF Seiten importieren. Anschliessend kann die Zeichnung als SVG gespeichert werden. Es ist wichtig eine aktuelle Version von Inkscape zu benützen.

Es wird angeraten, das erstellte SVG dahingehend zu überprüfen, dass das Root-Element das Attribut `viewBox` enthält, welches sicherstellt, dass die SVG Bilder korrekt skaliert werden.

[5] http://inkscape.org

Inkscape unterstützt zur Zeit keine SVG 1.2 Funktionalität, insbesondere was Spezialfarben angeht. Inkscape operiert momentan ausschliesslich im sRGB Farbraum. Sämtliche Farbwerte im PDF werden nach sRGB konvertiert. Falls im PDF z.B. Schmuckfarben verwendet werden und diese erhalten bleiben sollen, muss das SVG manuell optimiert werden und mit den Farb-Features von SVG 1.2 ergänzt werden.

Glossar

A

Apache Derby

Apache Derby[6] ist eine in Java geschriebene Embedded Datenbank unter der Apache Lizenz Version 2.0. Sie wird im Rahmen des "DB" Projektes der Apache Software Foundation[7] entwickelt.

D

Dezentraler Druck

Unter dezentralem Druck verstehen wir den Druck auf verschiedenste, dezentral aufgestellte Arbeitsplatz- und Abteilungsdrucker. Es ist zu erwarten, dass der Maschinenpark recht heterogen sein kann, weshalb u.U. verschiedene Druckprotokolle und -formate verwendet werden müssen. Weiter steht man hier vor der Herausforderung alle Drucker zentral verwalten zu können und bestimmte Drucker geographisch gesehen nahen Benutzern zuordnen zu können.

Derby

Siehe Apache Derby.

J

Job Control Language

Unter Job Control Languages (JCL) verstehen wir im Zusammenhang von WOD Sprachen zur Druckersteuerung. Die wichtigste ist PJL.

P

PostgresQL

PostgresQL[8] ist eine verbreitete, relationale Datenbank mit über 15 Jahren Entwicklungshistorie. Sie ist Open Source und unter einer BSD-ähnlichen Lizenz erhältlich.

Print Job Language

PJL ist eine von Hewlett-Packard entwickelte Sprache zur Umschaltung der Druckersprache, zur Steuerung der Druckereinstellungen und zur Abfrage des Druckerstatus. PJL wird von den meisten Druckern

[6] http://db.apache.org/derby/
[7] http://www.apache.org
[8] http://www.postgresql.org/

unterstützt, insbesondere von denen, welche die JetDirect (oder RAW) Schnittstelle auf Port 9100 unterstützen, über welchen u.a. Status-Abfragen getätigt werden können. Manche Drucker erkennen automatisch, um welche Druckersprache es sich handelt und schalten dann entsprechend um. Mit PJL ist es aber neben der expliziten Umschaltung der Druckersprache auch möglich, gewisse Einstellungen vorzunehmen.

Beispiel 108. Beispiel eines PJL Intros für einen PostScript Job

```
<ESC>%-12345X@PJL SET RESOLUTION=300
@PJL ENTER LANGUAGE=POSTSCRIPT
%!PS-Adobe-3.0
%%LanguageLevel: 2
etc. etc.
```

U

Universal Resource Identifier

Eine URI ist eine Zeichenkette, die eine bestimmte Ressource (abstrakt oder physisch) eindeutig identifiziert. Die Syntax ist in RFC 1630[9] definiert.

Z

Zentraler Druck

Unter zentralem Druck verstehen wir den Druck in einem Druck-Zentrum, wo wenige, grosse Druckmaschinen und ev. Verpackungsmaschinen betrieben werden. Die Druckjobs werden hierbei meist von Mitarbeitern des Druck-Zentrums auf freie Drucker verteilt. Bei der Erstellung des Druckdatenstroms ist also selten bekannt, auf welchem Drucker schlussendlich gedruckt wird.

[9] http://tools.ietf.org/html/rfc1630

Stichwortverzeichnis

K

L

M

O

P

R

S

T

U

V

W

X